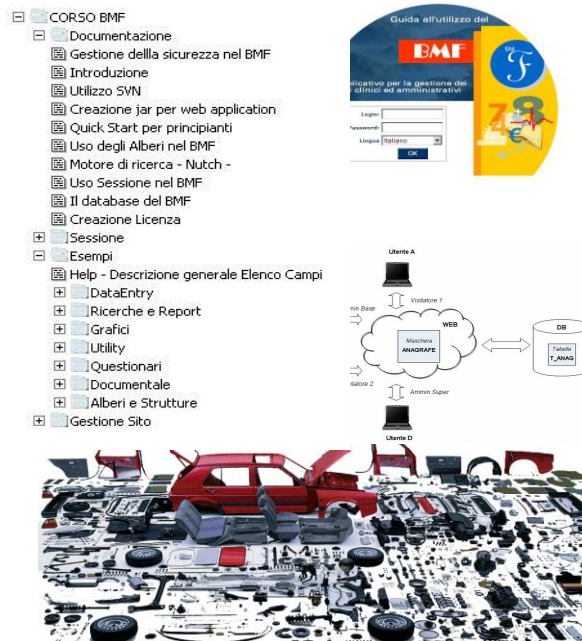


Maurizio Mangione, Gianna Alberini, Giorgia Vivoli

Il framework BMF 3

per lo sviluppo di applicazioni ICT in Life Science con esercitazioni e progetto pilota



Sommario

1.	Introduzione.....	4
1.1	Requisiti HW/SW	5
1.2	Le esigenze funzionali	Errore. Il segnalibro non è definito.
1.3	Oggetti astratti e template	Errore. Il segnalibro non è definito.
1.4	Login e sessione	Errore. Il segnalibro non è definito.
2.	Architettura del framework BMF3	12
2.1	Server e Client Side.....	12
2.2	Librerie open source	14
3.	Il database.....	17
3.1	Creazione e naming di una generica tabella	17
3.2	Naming degli altri oggetti del DB	18
3.3	Tabelle del BMF	19
3.4	Modello Concettuale: “la gestione degli oggetti e dei profili”	20
3.5	Modello Concettuale: “la gestione degli alberi”	21
4.	Configurazione Sito	23
4.1	Gestione Oggetti.....	23
4.2	Configurazione oggetti	25
4.3	Oggetti per profilo	26
4.4	Navigazione	27
4.5	Sicurezza Profilazione Utenti Locali	28
5.	Oggetti astratti	32
5.1	Oggetti per la visualizzazione	32
5.1.1	Report	32
5.1.2	Approfondimento - Visualizzazione per colonna	36
5.1.3	Chart.....	38
5.1.4	Calendar	46
5.2	Oggetti per il data entry	48
5.2.1	Input-form	48
5.2.2	Tipologie di text-field	54
5.2.3	Elementi separatori	58
5.2.4	Pulsante info su un campo	59
5.2.5	Validazione dei campi.....	60
5.2.6	Campi correlati.....	62
5.2.7	Inserimento multiplo.....	63
5.2.8	Data entry con storicizzazione.....	65
5.2.9	Data entry per riga	68
5.3	Oggetti per la ricerca	71
5.3.1	Filter	71
5.4	Oggetti per la navigazione	74

5.4.1	Menu item e menu folder	74
5.4.2	Button e Group button.....	76
5.5	Altri oggetti BMF.....	78
5.5.1	Action	78
5.5.2	Carousel	79
5.5.3	Period	81
5.5.4	Stored Procedure	82
5.5.5	Select.....	83
5.6	Opzioni avanzate.....	84
5.6.1	Record selezionabili: report annidati e associati ad input-form..	84
5.6.2	Composizione link	86
5.6.3	Composizione della query string e gestione dei sottotitoli	89
6.	Personalizzare la web-app	91
6.1	Filtro personalizzato	92
6.2	Input-form personalizzato	93
6.3	Report personalizzato	94
6.4	Altre Personalizzazioni.....	94
7.	Upload e download file	95
8.	Alberi.....	97
8.1	Strutture	98
8.2	Radici	101
8.2.1	Definizione Radici	101
8.2.2	Strutture Radici	102
8.3	Tipi.....	102
8.4	Navigazione Gerarchie	103
8.5	Associa Link a Struttura	104
8.6	Controlli	106
8.7	Sicurezza	107
8.8	Database e Viste V_ALBERO_*	109
8.9	Oggetti Tree	110

1. Introduzione

BMFramework (BMF) è un tool di sviluppo software che permette di realizzare *applicazioni web dinamiche*, ossia applicazioni costruite su uno o più database relazionali per la memorizzazione e la gestione dei dati di interesse.

Attraverso le pagine web di tali applicazioni sarà possibile effettuare operazioni di: ricerca, visualizzazione e data entry sulle tabelle del database, per una completa gestione dei dati da parte dell'utente finale.

Queste operazioni vengono eseguite tramite interfaccia web utilizzando un comune browser (Chrome, Safari, Internet Explorer, Mozilla Firefox, ecc.).

BMF è un tool di sviluppo nato per facilitare il lavoro dello sviluppatore di software perché, per creare un nuovo applicativo, non è necessario saper programmare con uno specifico linguaggio ad oggetti (Java, C++, ecc.) ma è sufficiente conoscere in maniera approfondita il linguaggio **SQL** e le **funzionalità del tool** per poter ottenere rapidamente le maschere di gestione e visualizzazione dei dati, i pulsanti associati, eventuali rappresentazioni grafiche dei dati, e gli altri oggetti tipici di un sito web dinamico.

Nel tempo si è modificata l'architettura permettendo anche a programmatori esperti di lavorare con Javascript.

Anche i *controlli di integrità* sui dati inseriti sono demandati al BMF, che li implementa *lato client*: utilizzando codice **Javascript standard**; *lato server*: utilizzando il naming richiesto e il linguaggio DDL del database.

In questo capitolo introdurremo le regole che gli elementi di un database BMF devono rispettare, relative in particolare alle convenzioni sui nomi (**naming**) di tabelle, campi, relazioni, viste, ecc. Pertanto, partiremo dall'ipotesi di un database relazionale correttamente progettato dal punto di vista delle entità e delle relazioni che rispetti tali regole, e seguiremo un esempio completo di implementazione di un sito con il BMF. Tra le varie funzionalità implementate che andremo ad analizzare, studiare e provare merita un'introduzione particolare la gestione delle utenze. Questo perché in ambito e-health le utenze hanno molti vincoli dettati dalla normativa vigente. Il BMF prevede un meccanismo di autenticazione che comprende la gestione classica a profili; permette quindi di assegnare a ciascun utente un *profilo* costituito da un insieme di attributi quali, ad esempio, *funzione, ruolo, posizione, sede, ecc.* all'interno di un'organizzazione aziendale. Ad ogni profilo è assegnato un livello di visibilità sui dati, per cui è possibile implementare un controllo completo sulle operazioni realizzabili da ciascun utente finale. Inoltre, per le applicazioni che gestiscono dati sensibili si può configurare l'accesso ai dati anche per singola login (ad esempio i medici che hanno ottenuto il consenso informato dai pazienti in cura presso la struttura in cui esercitano).

1.1 Requisiti HW/SW

Per creare applicazioni attraverso l'utilizzo del BMF è necessario predisporre una piattaforma a **tre livelli** (*Web-Application-Database Server*) in grado di eseguire **codice Java**. (in stage va bene anche a 2 livelli: Appl. e Db)

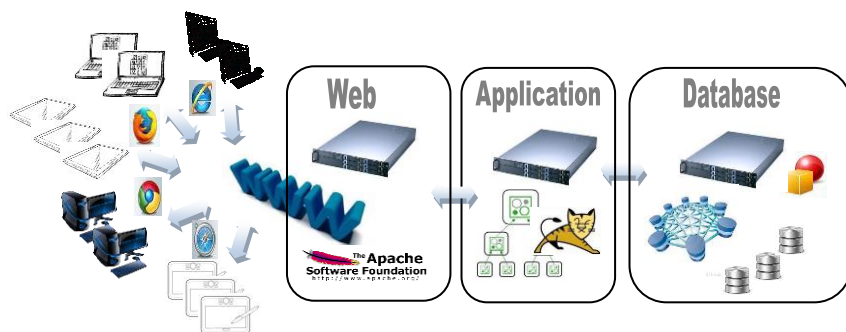


Figura 1 Esempio di architettura 3 livelli con web,application e db server

Una possibile configurazione della piattaforma è data dalla combinazione dei seguenti elementi, a seconda della loro compatibilità:

- **Sistemi operativi:** Unix, Linux, Windows, OSX, etc.
- **Web server:** Apache, IIS, etc.
- **Application server:** Tomcat, Jrun, etc.
- **Basi di dati:** Oracle, DB2, MySQL, MS-SQLserver, Access
- **Client-Browser:** Internet Explorer, Mozilla FireFox, Google Chrome, etc.

Relativamente ai browser è stata osservata una maggiore affidabilità utilizzando Firefox. Se necessario si può disporre anche di una licenza BMF per ogni server installato, indipendentemente dal numero delle postazioni utente (PC in cui si utilizzerà l'applicativo finale per effettuare operazioni di interrogazione e data entry). Esiste una versione del BMF, adeguatamente configurata per le applicazioni didattiche universitarie (per gli studenti).

BMF è un software Open Source con licenza LGPL.

1.2 Le esigenze funzionali

Di seguito analizzeremo alcune classiche funzionalità offerte dai sistemi web su database, che soddisfano le esigenze di diverse tipologie di utenti: inserimento dei dati, visualizzazione e stampa, interrogazioni mediante filtri di ricerca, ecc; spiegheremo, quindi, come tali esigenze vengono interpretate e realizzate da BMF. In qualunque ambiente applicativo si operi, è solitamente richiesta una diversificazione degli accessi alla stessa risorsa fisica, in base alle varie tipologie di utenti interessati; una rappresentazione schematica è mostrata nella figura seguente:

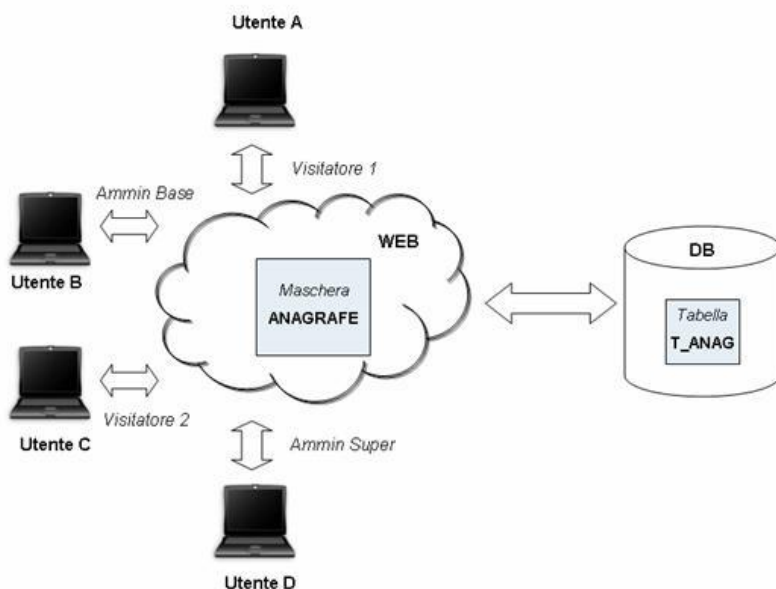


Figura 2– Accessi differenziati alla tabella T_ANAG, mediante la maschera ANAGRAFE.

Nel database di esempio è definita la tabella **T_ANAG**, che raccoglie un certo numero di informazioni di tipo anagrafico; a livello di applicativo web, si vuole mostrare questa risorsa attraverso la maschera **ANAGRAFE**, che verrà visualizzata con caratteristiche differenti a seconda dell'utente loggato e del profilo associato (*Visitatore 1*, *Ammin Base*, ecc.)

Ciascun utente, infatti, viene riconosciuto dal sistema mediante la propria **login** e **password** ed il relativo **profilo** associato.

Gli utenti a cui viene attribuito il profilo **Visitatore 1** e **Visitatore 2** hanno possibilità di lettura dei dati (tutti i campi o alcuni campi di T_ANAG); **Ammin Base** è il profilo che consente solo alcune funzionalità di lettura e scrittura;

Ammin Super può leggere e scrivere tutti i campi della tabella T_ANAG. La tabella che segue, schematizza questa ipotesi di accessi diversificati alla T_ANAG:

Utente	Login / Password	Profilo	Funzioni
A	VISIT_1/XXX	Visitatore 1	Accede a tutti i campi della tabella T_ANAG in sola lettura (R)
B	AMM_B/YYY	Amministratore Base	Accede ad alcuni campi della tabella T_ANAG in lettura/scrittura (R/W)
C	VISIT_2/ZZZ	Visitatore 2	Accede ad alcuni campi della tabella T_ANAG in sola lettura (R)
D	AMM_S/KKK	Amministratore Super	Accede a tutti i campi della tabella T_ANAG lettura/scrittura (R/W)

1.3 Oggetti astratti, template e maschere

Per rappresentare quanto necessario alla realizzazione delle quattro diverse funzionalità secondo la logica di **BMF**, ci serviremo di alcuni **oggetti astratti**:

1. Oggetti astratti di tipo PULSANTE **P_ANAG**
2. Oggetti astratti di tipo DATA ENTRY **T_ANAG**
3. Oggetti astratti di tipo VISUALIZZAZIONE **V_ANAG**

Ogni oggetto astratto implementa una specifica funzionalità, in base alla quale ad ognuno di essi è associato uno specifico **template**, ovvero un modello di presentazione di tale funzionalità all'utente.

In particolare:

- Tutti gli utenti avranno a disposizione un pulsante **P_ANAG** attraverso il quale richiamare la visualizzazione della maschera ANAGRAFE;
- L'Utente A e l'Utente C avranno a disposizione un oggetto **V_ANAG** per la visualizzazione dei dati;
- L'Utente B e l'utente D avranno a disposizione sia un oggetto **V_ANAG**, per la visualizzazione dei dati, sia un oggetto **T_ANAG**, per l'aggiornamento degli stessi.

Poiché in due casi (Utente A e Utente C) è prevista la sola visualizzazione dei dati, la relativa maschera ANAGRAFE sarà composta dal solo oggetto astratto V_ANAG. Negli altri due casi (Utente B e Utente D) la maschera invocata dal pulsante P_ANAG dovrà essere costituita da due oggetti, V_ANAG e T_ANAG, per l'esecuzione delle due operazioni: visualizzazione e modifica dei dati della tabella in questione.

Dunque, l'azione associata all'oggetto astratto **P_ANAG** (pulsante) degli utenti A e C sarà diversa da quella impostata per gli utenti D e B, proprio per questa differente composizione delle maschere ANAGRAFE invocate dal pulsante stesso.

Di seguito, una rappresentazione della struttura delle maschere ANAGRAFE per le quattro tipologie di utenti, e la corrispondente **codifica** degli oggetti astratti utilizzati dal BMF (Figura 3).

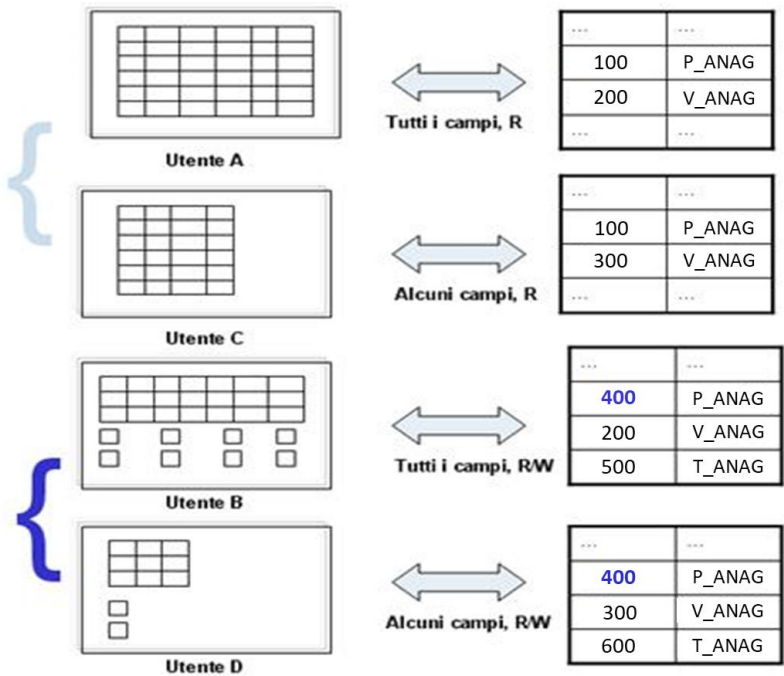


Figura 3 Maschere, oggetti astratti e codifica in BMF

Analizziamo la parte sinistra della Figura 3.

La maschera ANAGRAFE per gli Utenti visitatori A e C, che possono solo visualizzare i dati di T_ANAG (*R=Read*), avrà la stessa struttura: sarà una maschera di *visualizzazione*, composta dal solo oggetto astratto V_ANAG; l'unica differenza tra i due Utenti riguarderà i dati visualizzati (per l'Utente A tutti i campi, per l'Utente C solo alcuni campi di T_ANAG).

La maschera ANAGRAFE per gli Utenti amministratori B e D, che possono visualizzare/modificare i dati di T_ANAG (*R/W=Read/Write*), sarà costituita anche da un oggetto astratto con *template di data entry* (T_ANAG), differenziandosi soltanto per il numero di campi mostrati: per l'Utente B tutti i campi in visualizzazione/modifica, per l'Utente D solo alcuni campi in visualizzazione/modifica di T_ANAG.

Nella parte destra di Figura 3 viene rappresentata la codifica di queste associazioni in ambiente **BMF**.

UTENTE A

Avrà associato un oggetto **P_ANAG** di tipo pulsante, per attivare la visualizzazione dei dati; inoltre, avrà associato **V_ANAG**, oggetto di tipo visualizzazione di tutti i campi della tabella in questione.

UTENTE C

Avrà associato un oggetto **P_ANAG** di tipo pulsante, per attivare la visualizzazione dei dati; inoltre, **V_ANAG**, oggetto di tipo visualizzazione di alcuni campi della tabella in questione. Si osservi che il codice dell'oggetto pulsante **P_ANAG** è lo stesso per i due utenti A e C (Codice **100**) poiché in entrambi i casi dovrà invocare una maschera costituita da un solo oggetto astratto di visualizzazione. Tale oggetto, **V_ANAG**, avrà un codice diverso (200 per l'Utente A, 300 per l'Utente C), poiché l'Utente A è abilitato alla visualizzazione di tutti i campi della tabella T_ANAG, mentre l'Utente C solo di alcuni.

UTENTE B

Avrà associato un oggetto **P_ANAG** di tipo pulsante, per attivare la gestione in R/W dei dati; inoltre, avrà associati **V_ANAG**, oggetto di tipo visualizzazione di tutti i campi della tabella, e **T_ANAG**, oggetto di tipo data entry di tutti i campi della tabella.

UTENTE D

Avrà associato un oggetto **P_ANAG** di tipo pulsante, per attivare la gestione in R/W dei dati; inoltre, **V_ANAG**, oggetto di tipo visualizzazione di alcuni campi della tabella, e **T_ANAG**, oggetto di tipo data entry di alcuni campi della tabella. Anche in questo caso, si osservi che il codice dell'oggetto pulsante **P_ANAG** è lo stesso per i due utenti B e D (Codice **400**) poiché in entrambi i casi dovrà invocare una maschera ANAGRAFE per la gestione in R/W dei dati, quindi costituita da un

oggetto astratto di visualizzazione e uno per l'operazione di data-entry. L'oggetto **T_ANAG** avrà invece un codice diverso (500 per l'Utente B, 600 per l'Utente D), poiché l'Utente B è abilitato alla scrittura di tutti i campi della T_ANAG, mentre l'Utente D solo di alcuni campi. Per la stessa ragione, l'oggetto **V_ANAG** avrà un codice diverso (200 per l'Utente B, 300 per l'Utente D), poiché l'Utente B è abilitato alla lettura di tutti i campi della T_ANAG, mentre l'Utente D solo di alcuni campi.

Il **BMF** memorizza queste informazioni per tutti gli utenti del sistema nella tabella **T_OBJECT** ed in altre tabelle ad essa correlate, come spiegheremo più avanti.

Al login dell'Utente A, solo le informazioni riguardanti A vengono memorizzate nella **SESSIONE**, area di memoria dell'application server che contiene tutte le informazioni relative allo stato dell'utente nella sessione corrente; tale area, che nasce al login, viene ripulita con il logout dall'applicazione, con la chiusura del browser, oppure in seguito al time-out sulla connessione.

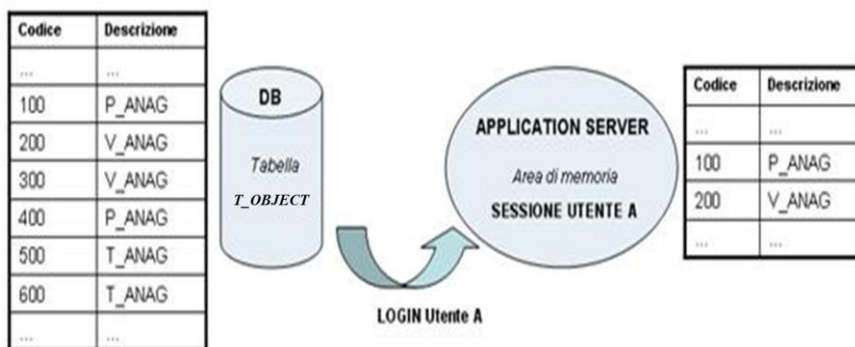


Figura 4 – Login: memorizzazione degli oggetti associati all'Utente A nella SESSIONE

BMF utilizza la SESSIONE per visualizzare e rendere disponibili tutti e soli gli oggetti di pertinenza dell'utente autenticato, in base al suo profilo.

1.4 Login e sessione

La figura che segue mostra più in dettaglio la logica funzionale che interviene quando un generico Utente U1 si collega ad un sistema web realizzato con BMF.

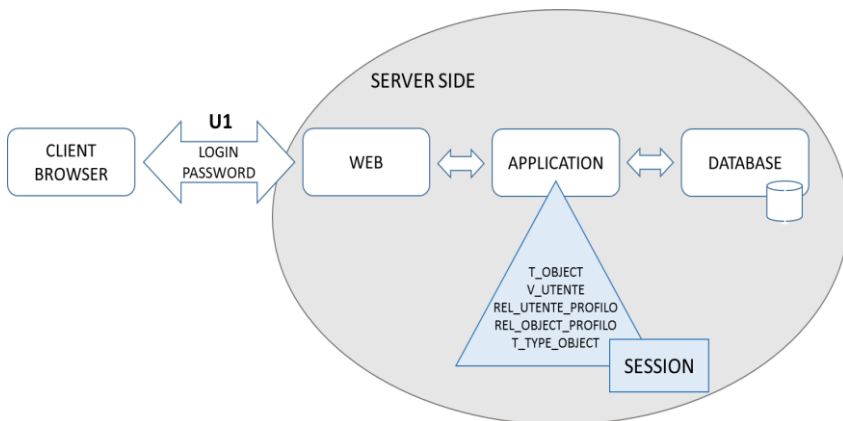


Figura 5 - Login dell'utente U1 e memorizzazione dei dati relativi in Sessione U1

Il client a cui è collegato U1 ha attivo un browser (ad esempio Internet Explorer, I.E.). Dopo aver digitato l'indirizzo del sito web desiderato, ed aver impostato login e password, la richiesta passa attraverso la rete e raggiunge il server che immaginiamo, per semplicità, essere un'unica macchina su cui risiedono i tre livelli: **Web**, **Application** e **DB Server**.

La parte Web del server accoglie la richiesta del client e la trasferisce all'Application Server. A questo punto la parte Application chiede al DB se l'utente è abilitato ad accedere al sistema, ovvero se è codificato come utente, mediante il controllo sulla login e password immesse.

In caso affermativo, la parte DB passa all'Application l'intera configurazione dell'utente, che verrà memorizzata in **SESSIONE**.

In particolare, in questa area di memoria verranno memorizzate le seguenti informazioni (reperite da alcune tabelle del BMF, indicate fra parentesi):

- Gli oggetti astratti associati all'utente U1 ed il loro tipo (tabelle **T_OBJECT**, **T_TYPE_OBJECT**);
- Gli attributi quali: ruolo, funzione, categoria, ed altre caratteristiche che riguardano l'utente come personale (**V_UTENTE**);
- Il suo profilo (**REL_UTENTE_PROFILO**);
- Le relazioni esistenti fra gli oggetti ed il profilo utente (**REL_OBJECT_PROFILO**).

2. Architettura del framework BMF3

L'architettura del framework BMF3 presenta una netta separazione tra la parte server e quella client, per ottimizzare i tempi di risposta ed avere una parte client facilmente personalizzabile.

2.1 Server e Client Side

La parte **server** è basata su un'architettura a tre livelli in grado di eseguire codice Java. I tre livelli su cui si articola la suddetta architettura sono:

- il livello Web Server
- il livello Application Server
- il livello Database Server

Le richieste della parte client arrivano al Web Server, attraverso protocollo HTTP o HTTPS. Il Web Server, tramite protocollo TCP IP, comunica con l'Application Server. Nel caso del BMF, le precedenti due funzioni sono svolte dall'Apache Tomcat, comunemente conosciuto come Tomcat, un software java dell'Apache Software Foundation. Il Tomcat elabora la richiesta del client tramite codice Java, quindi genera una nuova richiesta in formato PLSQL che invia al DB Server. Quest'ultimo elabora la richiesta ed eventualmente risponde in PLSQ dando il via al percorso inverso. Le Web Services RESTEasy, tramite il Tomcat, comunicano con la parte client inviando risposte in formato JSON. Il formato JSON è stato scelto in quanto è facilmente utilizzabile da qualsiasi linguaggio di programmazione e consente di mantenere la separazione logica tra il client e il server così da poter cambiare interamente all'occorrenza l'implementazione client side senza dover intervenire su quella server e viceversa.

Per la parte di configurazione e accesso alla base di dati, invece, viene utilizzata la libreria Spring.

Il **client** si frappa tra l'utente e il server: è attraverso di esso che l'utente può accedere ai servizi e alle risorse messe a disposizione dal server. Al Client, in altre parole, è demandata la gestione della visualizzazione dei dati e delle pagine attraverso le quali interfacciarsi ai dati stessi.

Lo schema architetturale implementato per costruire l'interfaccia utente è il Model-View-Presenter il quale contribuisce a mantenere la separazione client-server di cui sopra. Tale architettura è costituita da tre elementi:

- **Model:** definisce i dati da visualizzare all'interno dell'interfaccia;
- **View:** interfaccia "passiva" che mostra i dati e che indirizza i comandi dell'utente (gli eventi), le operazioni da effettuare sui dati al presenter;
- **Presenter:** agisce tra il model e il view. Recupera i dati dal model e formatta per visualizzarli nella view.

Tale architettura fornisce agli sviluppatori un modello per creare un'applicazione flessibile, ovvero che consente di modificare o aggiungere funzionalità agendo solo su uno specifico livello, e scalabile, ovvero per la quale è possibile aggiungere ulteriori funzionalità senza doverne modificare le caratteristiche fondamentali. Una qualunque richiesta dell'utente è effettuata attraverso la Graphical User Interface (GUI) presentata all'interno di una pagina web alla quale si accede col-legandosi a un Browser. Tale richiesta viene elaborata direttamente lato client aggiornando solo porzioni della pagina web in background senza la necessità di dover richiedere un ricaricamento integrale da server. Questo è possibile perché i dati vengono recuperati dal server tramite delle chiamate asincrone (AJAX) senza fare redirect per visualizzare i risultati delle elaborazioni.

La parte client del BMF è completamente realizzata in JavaScript mediante il supporto di framework quali:

- jQuery <https://jquery.com/>
- Backbone.js <http://backbonejs.org/>
- Marionette.js <https://marionettejs.com/>
- Underscore.js <https://underscorejs.org/>
- Handlebars.js <http://handlebarsjs.com/>
- Require.js <https://requirejs.org/>

I template sono realizzati in HTML5 e la parte grafica è stata sviluppata utilizzando Bootstrap, front-end framework per la progettazione di interfacce web: si tratta di una raccolta di strumenti per la creazione di siti e web-application, contenente modelli di progettazione basati su HTML e CSS, sia per la tipografia sia per le varie componenti dell'interfaccia, come moduli, pulsanti e navigazione.

Di seguito sarà presentata una sintetica descrizione degli strumenti sopra elencati, rimandando al lettore un loro approfondimento più dettagliato.



Figura 6 - Framework utilizzati per lo sviluppo della parte client del BMF3

2.2 Librerie open source

jQuery

jQuery è una libreria nata con lo scopo di rendere più sintetico il linguaggio JavaScript, di semplificare la gestione degli eventi e l'animazione di elementi DOM in pagine HTML, e di implementare funzionalità AJAX per comunicazioni con il server: il framework, dunque, fornisce metodi per la manipolazione semplificata e standardizzata del DOM e per il controllo dello stile CSS degli elementi; la gestione delle chiamate asincrone è semplificata, e sono fornite le funzioni per caricare contenuti, eseguire richieste asincrone (con metodo GET/POST), caricare oggetti Json e file .js remoti.

L'oggetto principale, di nome jQuery, è genericamente utilizzato tramite il suo alias **\$**.

Si consideri il seguente esempio:

```
document.getElementById("mioLink").href; // JavaScript nativo
```

Questa istruzione JavaScript seleziona, nel documento HTML, l'elemento con ID "mioLink" e accede al suo attributo href. L'equivalente jQuery di questa istruzione è:

```
$("#mioLink").attr("href"); // jQuery
```

Questo semplice esempio evidenzia alcune caratteristiche fondamentali di jQuery: la brevità del codice, la sua intuitività ed il fatto che, per ottenere l'accesso ad un elemento HTML della pagina, questo venga passato alla funzione **\$()** mediante un selettore che ha la stessa sintassi di quelli CSS, ovvero in base al suo id, alla sua classe, ecc.

Un'altra caratteristica da sottolineare è la possibile concatenazione tra i metodi di jQuery, che rende la scrittura e la lettura del codice più lineare:

```
$("#mioLink").text("Nuovo testo").css("color","red"); //modifica del testo e del colore del link
```

Per utilizzare jQuery è sufficiente scaricare la libreria e collegarla alla pagina HTML, inserendo nel tag head il seguente codice:

```
<script language="JavaScript" type="text/JavaScript" src="jquery-1.3.2.min.js"></script>
```

Backbone.js e Marionette.js

Backbone.js è un framework JavaScript il quale fornisce strutture al fine di migliorare l'organizzazione del codice e garantire così scalabilità e manutenibilità all'applicazione che si sta sviluppando.

Per la sua architettura, Backbone.js rientra nella categoria delle librerie **MV***; infatti, i suoi componenti base sono:

- **Model**, oggetto discreto contenente una serie di dati sotto forma di attributi;
- **Collection**, oggetto contenente una raccolta di modelli dello stesso tipo, attraverso il quale è possibile ordinare, filtrare e manipolare i modelli contenuti;
- **View**, componente che funge da tramite tra interfaccia e modelli, definendone la logica di interazione ed inviando gli input al modello. In un'applicazione Backbone.js, l'interazione modifica prima lo stato di un model scatenando quindi la reazione della view che si aggiorna di conseguenza;
- **Router**, componetne per il routing e la gestione centralizzata dello stato dell'applicazione; è possibile realizzare single page app a partire da questo componente.

Marionette.js è un libreria JavaScript che estende il framework Backbone.js mettendo a disposizione view e soluzioni architetturali che semplificano il codice. Per il suo funzionamento, Backbone richiede Underscore.js (hard dependency) e jQuery (soft dependency).

Underscore.js

Underscore.js è una libreria JavaScript che aggiunge una serie di funzioni e metodi di utilità agli oggetti built-in di JavaScript, per attività di programmazione comuni come la manipolazione di Array, Oggetti e Collezioni, che consentono di risparmiare moltissime righe di codice (ad esempio each, find, filter, where, contains, ecc).

Handlebars.js

Handlebars è un **template engine** JavaScript. Per tradurre oggetti Json in HTML si fa spesso ricorso ai template engine, script che consentono la definizione di un blocco HTML (detto appunto template) nel quale identificare dei segnaposto (placeholders) da sostituire con i dati da visualizzare.

Require.js

Require.js è un **module loader** open source, che permette il caricamento di moduli JavaScript, al bisogno e nel giusto ordine, e la gestione delle loro

dipendenze, funzionalità indispensabile nel caso di progetti di grandi dimensioni: ogni script, dunque, potrà condividere le proprie funzionalità con altri moduli, grazie alla cosiddetta iniezione di dipendenza. Tale condivisione si unisce ad altri vantaggi, quali la gestione dell'ordine di caricamento degli script, l'assenza di duplicazioni delle inclusioni, l'ottimizzazione della memoria per pagine web più veloci.

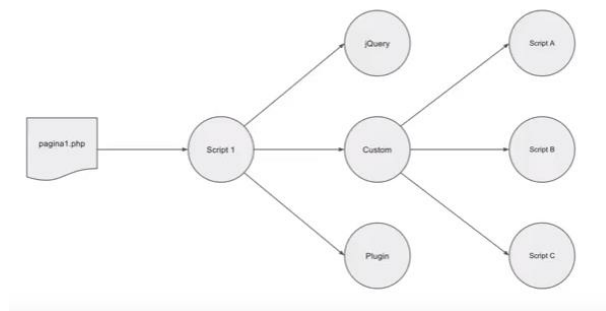


Figura 7 – Require.js: dipendenza e condivisione di funzionalità

Json

Json (JavaScript Object Notation) è un formato per l'interscambio di dati in applicazioni client – server.

Nonostante nasca da JavaScript, Json è un formato testuale completamente indipendente dal linguaggio, perciò può essere utilizzato ed interpretato correttamente anche da altri linguaggi come PHP, Java, C, Python e molti altri.

I dati sono rappresentati sotto forma di coppie nome-valore separate mediante il simbolo dei due punti (:) che funge da operatore di assegnazione.

I tipi di dati supportati dal formato Json sono:

- booleani (true e false),
- numeri (interi, reali, virgola mobile),
- stringhe,
- array (sequenze ordinate di valori, separati da virgole e racchiusi in parentesi quadre []),
- oggetti (sequenze non ordinate di coppie chiave-valore racchiuse in parentesi graffe),
- null.

Di seguito, un esempio di un file JSON che descrive una persona:

```
{"nome": "Mario", "cognome": "Rossi",  
  "nascita": {"giorno": "1", "mese": "1", "anno": "1980" }}
```


3. Il database

Il database del BMF si compone di un insieme di tabelle, di norma create nello stesso *schema* del DB dell'applicativo.

Tutte le tabelle, sia quelle proprie di BMF sia quelle dell'applicazione, devono essere definite rispettando delle particolari *convenzioni sui nomi* o **regole di naming** che descriveremo di seguito, per poi passare ad analizzare in dettaglio le tabelle del DB che compongono il BMF.

3.1 Creazione e naming di una generica tabella

In fase di creazione di una tabella, sia nel caso di tabella propria dell'applicativo sia nel caso di tabella del BMF, dovranno essere rispettate le seguenti convenzioni sui nomi:

1. Nome della tabella, da definire come T_<NOME_TABELLA>

Ad esempio T_COMMESSA

2. La Chiave primaria (*Primary key*) di tipo progressivo numerico deve avere il nome T_<NOME_TABELLA>_CODICE

Ad esempio T_COMMESSA_CODICE

3. La Chiave primaria deve essere definita come primo campo della tabella

4. La Chiave esterna (*Foreign key*) anch'essa di tipo numerico deve avere il nome T_<NOME_TABELLA_COLLEGATA>_CODICE

Ad esempio la tabella T_ALLEGATI presenta fra i diversi campi T_DOCUMENTI_CODICE che rappresenta la chiave esterna di collegamento alla tabella T_DOCUMENTI

5. Un generico campo è definito come

T_<NOME_TABELLA>_<NOME_CAMPO>

Ad esempio T_COMMESSA_ANNO

6. I Constraint di Primary key hanno come prefisso PK_

Ad esempio PK_T_COMMESSA definito sulla chiave primaria T_COMMESSA_CODICE

7. I Constraint di Foreign key hanno prefisso FK_

Ad esempio la tabella T_OGGETTO_LABEL, collegata alla T_OGGETTO ha un constraint di Foreign key FK_T_OGGETTO_LABEL_OGG che fa riferimento alla chiave

esterna T_OGGETTO_CODICE (campo che rappresenta la chiave primaria della tabella principale T_OGGETTO).

Il seguente è un esempio di script (su DB Oracle) di creazione della tabella T_COMMESSA:

```
CREATE TABLE T_COMMESSA  
(  
  T_COMMESSA_CODICE NUMBER(10) NOT NULL,  
  T_COMMESSA_NOME VARCHAR2(60) NOT NULL,  
  T_COMMESSA_DESC VARCHAR2(1000),  
  T_COMMESSA_DATA_INIZIO DATE,  
  T_COMMESSA_DATA_FINE DATE,  
  T_COMMESSA_NOTE VARCHAR2(2000),  
  T_COMMESSA_REFERENTE VARCHAR2(255),  
  T_COMMESSA_NEXT_REL DATE  
)  
CONSTRAINT PK_T_COMMESSA PRIMARY KEY (T_COMMESSA_CODICE);
```

3.2 Naming degli altri oggetti del DB

Di seguito, le altre convenzioni che gli *oggetti del database* devono rispettare per potersi correttamente interfacciare con BMF:

- REL_ Prefisso per una tabella di Relazione
Ad esempio REL_OGGETTO_PROFILO
- V_ Prefisso per una Vista
Ad esempio V_ATTRIBUTI_PERSONE
- P_ Prefisso per una Procedura
Ad esempio P_VALIDA_LOGIN
- TR_ Prefisso per un Trigger
Ad esempio TR_PERIODO
- S_ Prefisso per uno Snapshot
Ad esempio S_VALID_ATTIVITA_PREST_A

3.3 Tabelle del BMF

Di seguito lo schema fisico del database che rappresenta le tabelle proprie di BMF nella versione beta 0.1 del 2004 (Figura 8- Database BMF, modello fisico: prima release); queste tabelle coesistono nello stesso schema in cui vengono definite le tabelle dell'applicativo.

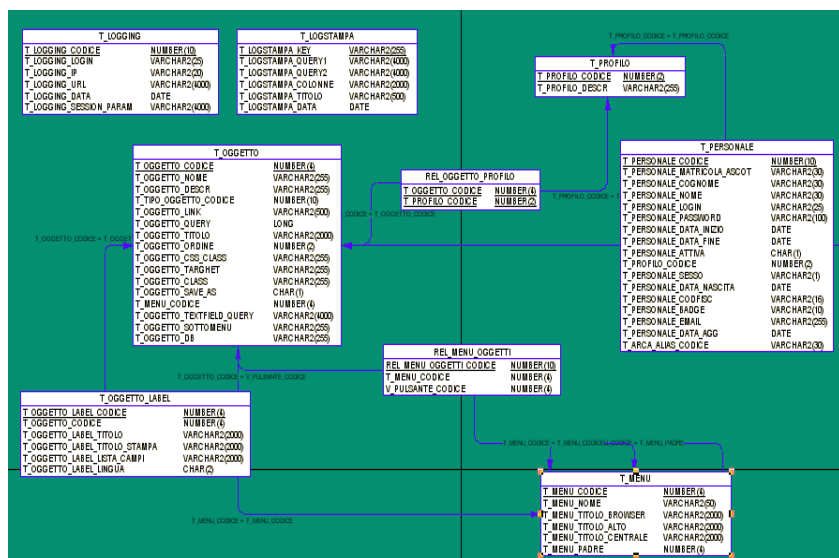


Figura 8- Database BMF, modello fisico: prima release

Dal 2004 ad oggi il BMF è stato utilizzato sia su progetti di ricerca che per sistemi web clinici, amministrativi e di governo. L'attuale release BMF 3.X è il frutto di anni di modifiche ed aggiornamenti, correzioni ed evoluzioni che hanno visto e vissuto l'evoluzione del Database.

Di seguito si presenterà il modello concettuale della versione BMF 3.X. Data la dimensione del database si rappresenterà il modello concettuale partizionato in “moduli funzionali”.

3.4 Modello Concettuale: “la gestione degli oggetti e dei profili”

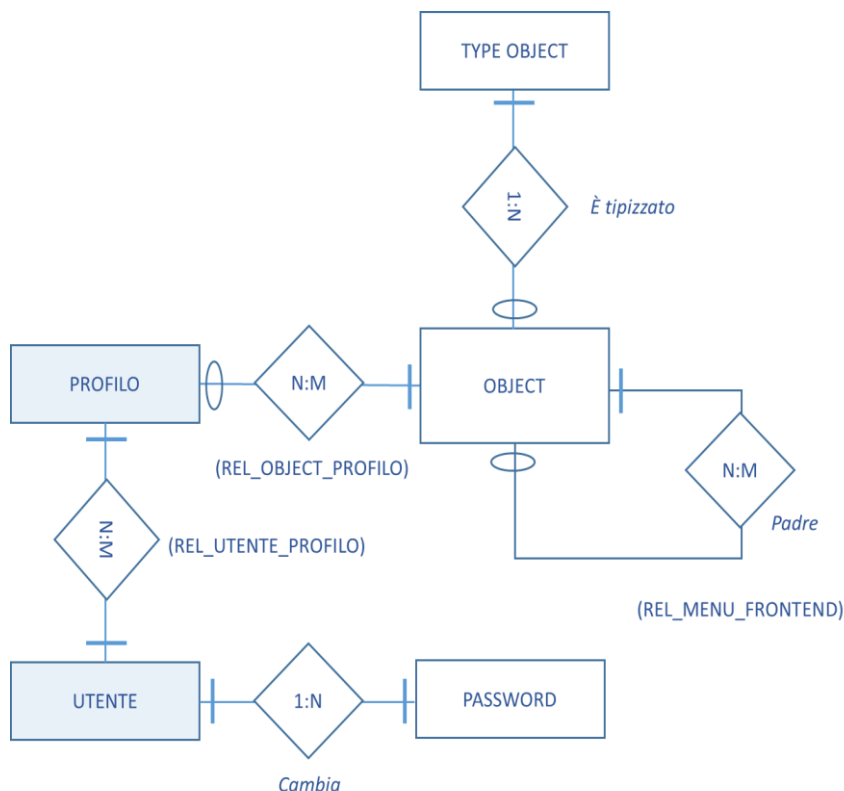


Figura 9 - Modello concettuale - oggetti e profili

T_OBJECT contiene tutti gli oggetti che costituiscono il front end, con le relative informazioni dichiarate in fase di configurazione (Codice, Nome, Descrizione, Query, Json, ecc.).

T_TYPE_OBJECT classifica gli oggetti del BMF in base alla loro tipologia (Report, Filter, Chart, ecc.);

T_PROFILO contiene i profili da associare agli utenti.

T_UTENTE contiene i nominativi degli utenti che possono accedere al sito; in essa si definiscono *login* e *password* di accesso.

T_PASSWORD contiene le password usate dagli utenti che risultano scadute. Viene utilizzata per la gestione delle password degli utenti in conformità con le direttive per la privacy.

REL_OBJECT_PROFILO contiene le associazioni tra oggetti e profili in modo da diversificare le gestioni da rendere accessibili, a seconda del profilo dell'utente.

REL_MENU_FRONTEND contiene le associazioni tra oggetti di tipo "menu folder" e oggetti di tipo "menu folder" o "menu item", per la costruzione della navigazione del sito. In questo modo viene costruita la struttura del menu.

REL_UTENTE_PROFILO associa ad ogni utente il profilo con cui accedere al sito.

3.5 Modello Concettuale: "la gestione degli alberi"

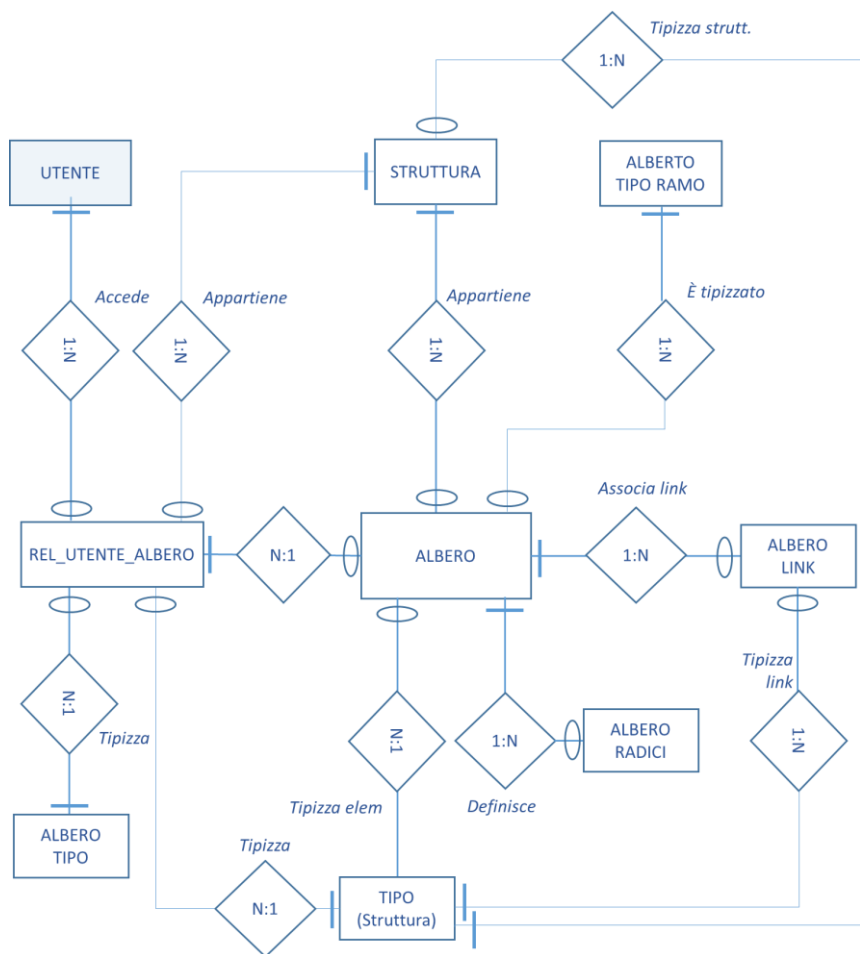


Figura 10 - Modello concettuale - alberi

Gli alberi sono delle strutture di navigazione che posso trovarsi negli applicativi. Spesso infatti si necessita di sviluppare navigazioni gerarchiche. Il BMF supporta

questo modello di navigazione dei dati, permettendo di realizzare interfacce ad albero. Perciò, è facile creare sia menu ad albero sia qualsiasi altra forma di navigazione gerarchica suggerita dai dati e diversificata per i vari profili applicativi. Per consentire la strutturazione dell'informazione e per ottenere una navigazione sicura si necessita delle seguenti tabelle:

- T_ALBERO_TIPO_RAMO, T_ALBERO_RADICI, T_ALBERO_TIPO sono tabelle dizionario necessarie per identificare componenti.
- T_STRUTTURA, T_TIPO servono a memorizzare tutti i nodi, radici e foglie che possono comporre l'albero.
- T_ALBERO, T_ALBERO_LINK per poter personalizzare l'albero definito nella struttura secondo le esigenze applicative.
- REL_UTENTE_ALBERO serve ad associare all'utente i nodi, radici, foglie dell'albero che può vedere.

4. Configurazione Sito

Il menù di configurazione del sito raccoglie le funzionalità (anch'esse sviluppate mediante BMF) che consentono al programmatore di sviluppare le interfacce dell'applicazione; esso è composto da 5 pulsanti: *Gestione Oggetti*, *Configura Oggetti*, *Navigazione Sito*, *Oggetti per Profilo*, *Sicurezza Profilazione Utenti Locali*.

4.1 Gestione Oggetti

Si tratta del cuore funzionale del sistema.

Attraverso questa funzionalità è possibile ricercare tutti gli oggetti dell'applicativo, modificarli, copiarli ed eliminarli; la maschera è composta da:

- un filtro, per la ricerca dell'oggetto di interesse;
- un report, per la visualizzazione dell'elenco degli oggetti risultanti dalla precedente ricerca;
- un form, per la modifica dei parametri di configurazione e per le operazioni di eliminazione e copia dell'oggetto selezionato, rappresentato in Figura 11- *Input-form di configurazione di un oggetto BMF*.

Copiando l'oggetto (operazione *save as*) sarà possibile agire direttamente sulla copia creata per modificarne alcuni campi ed ottenere rapidamente un nuovo oggetto.

Di seguito, l'elenco di tutti i parametri di configurazione presenti nel form e il loro significato:

- ID: codice identificativo dell'oggetto (progressivo numerico gestito direttamente dal BMF);
- Nome: nome dell'oggetto;
- Tipo: tipologia dell'oggetto, da scegliere tra Report, Chart, Calendar, Input-form, Filter, Menu Item, Menu Folder, Buttons, Action, Select, Period, Store Procedure, Carousel, Tree;
- Descrizione: nota descrittiva dell'oggetto;
- DB: il pool su cui si vuole eseguire la query (se questo campo viene lasciato vuoto, sarà utilizzato il DB di default del framework);
- Link: nel caso di oggetti di *menu item*, questo parametro rappresenta il link invocato;
- Link Params;

- Ordine: ordine di visualizzazione degli oggetti di tipo *menu folder* e *menu item* all'interno del menu laterale di navigazione;
- Query: query per il caricamento dei dati da DB;
- Json: oggetto Json di configurazione.


ID	<input type="text"/>
Nome	<input type="text"/>
Tipo	<input type="text"/> 
Descrizione	<input type="text"/>
DB	<input type="text"/>
Link	<input type="text"/>
Link Params	<input type="text"/>
Ordine	<input type="text"/>
Query	<div><div></div><div></div></div>
Json Config	<div><div></div><div></div></div>

Figura 11- Input-form di configurazione di un oggetto BMF

4.2 Configurazione oggetti

Si tratta di una funzionalità per la configurazione di nuovi oggetti o per la ricerca di un oggetto esistente al fine di modificarlo; il suo scopo è lo stesso della funzionalità *Gestione Oggetti*, precedentemente illustrata, ma in questo caso la fase di configurazione/modifica è scomposta in tre step separati ed è pensata per agevolare gli utenti meno esperti. Il primo step richiede la scelta dei parametri di configurazione: ID, Nome, Tipo, Descrizione, DB, Link, LinkParams, Ordine (Figura 12). Il secondo step è dedicato alla compilazione del Json file; tale compilazione potrà essere eseguita scegliendo tra due diverse opzioni:

- Configurazione per utenti esperti: edit libero, come previsto anche dalla funzionalità *Gestione Oggetti*;
- Configurazione standard, per gli utenti meno esperti: la maschera di compilazione e i relativi campi di inserimento cambiano a seconda della tipologia di oggetto, scelta nel primo step, e guidano l'utente nella configurazione. Il terzo ed ultimo step è dedicato alla scrittura della query per il caricamento dei dati da DB.

Figura 12 - Step 1: parametri di configurazione

4.3 Oggetti per profilo

Una volta creati tutti gli oggetti necessari al funzionamento della web-app, sarà necessario differenziare l'accesso agli stessi in relazione ai profili coinvolti.

Ogni profilo potrà accedere a determinati oggetti: questa associazione si realizza attraverso la funzionalità *Oggetti per Profilo*; la maschera invocata dal relativo pulsante si compone di:

- un *report*, per la visualizzazione di tutti i profili esistenti;
- un *form*, costituito dall'elenco *checkbox* di tutti gli oggetti che compongono il sito.

Selezionando un determinato profilo, gli oggetti ad esso assegnati saranno tutti quelli spuntati: sarà possibile assegnare altri oggetti semplicemente spuntando la relativa *checkbox* e, viceversa, revocare il diritto di accesso ad un oggetto rimuovendo la spunta.

La modalità *checkbox* permette associazioni multiple, cioè si possono selezionare più oggetti e legarli attraverso un'unica operazione ad un determinato profilo. Ogni oggetto creato viene associato automaticamente dal sistema al profilo *amministratore*, in modo da autorizzarlo alla visione di tutti gli oggetti senza che lo sviluppatore debba compiere manualmente questa associazione.

4.4 Navigazione

Si tratta della funzionalità che permette la costruzione del menu di navigazione del sito.

Ogni menu è costituito da pulsanti. Il BMF3 prevede la possibilità di creare due diverse tipologie di pulsanti: *menu folder* e *menu item*.

Gli oggetti di tipo *menu item* sono i veri e propri pulsanti, cliccando i quali viene invocato uno specifico link, associato loro in fase di configurazione.

Gli oggetti di tipo *menu folder*, invece, svolgono il ruolo di “contenitori”: raccolgono gruppi di pulsanti all’interno di menu gerarchici, e ad essi non è associato alcun link in fase di configurazione.

Attraverso la funzionalità Navigazione è possibile legare i pulsanti *menu item* ai rispettivi contenitori *menu folder* (o *menu folder* ad altri *menu folder*), mediante una maschera costituita da:

- un report, per la visualizzazione dei legami già creati;
- un semplice *form*, per la creazione di nuovi legami, nel qual sarà sufficiente specificare il *folder* (contenitore) e l’item (pulsante contenuto).

Si consideri il seguente esempio: si supponga di aver creato due oggetti *menu folder* (Menu1 e Menu2) e tre oggetti *menu item* (A, B, C) e di voler creare il menu-navigazione mostrato in Figura 13.

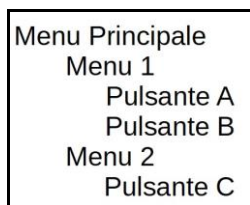


Figura 13 - Esempio di struttura di un menu di navigazione

Per fare ciò, utilizzando *form* all’interno della maschera Navigazione (Figura 14), sarà sufficiente dichiarare le seguenti associazioni:

- Folder: Menu Principale; Pulsante: Menu1;
- Folder: Menu Principale; Pulsante: Menu2;
- Folder: Menu Menu1; Pulsante: Pulsante A;
- Folder: Menu Menu1; Pulsante: Pulsante B;
- Folder: Menu Menu2; Pulsante: Pulsante C.

The image shows a web form for navigation configuration. It consists of three input fields: 'ID' (a text box), 'Folder' (a dropdown menu with 'Menu1' selected), and 'Pulsante' (a dropdown menu with 'PulsanteA' selected). Below these fields are three buttons: 'Clear', 'Salva' (with a save icon), and 'Elimina'.

Figura 14 - Form della maschera Navigazione

Si otterrà così il menu di navigazione laterale mostrato in **Figura 15**.

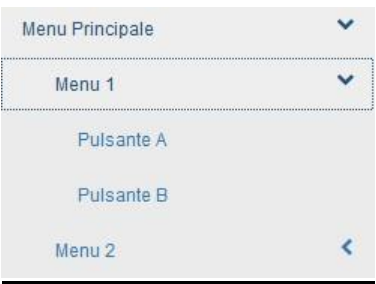


Figura 15 – Menu di navigazione ottenuto




4.5 Sicurezza Profilazione Utenti Locali

Sicurezza Profilazione Utenti Locali è un'opzione che raccoglie due diverse funzionalità: Profili e Utenti Locali.

Il pulsante **Profili** invoca una maschera costituita da un *report* per la visualizzazione di tutti i profili creati e un *form* per la modifica della descrizione associata al profilo (Figura 16).

Gestione Profili


Codice	Descrizione
0	AMMINISTRATORE DI SISTEMA
1	PROFILO TEST



Codice

Descrizione *

Clear

Salva 

Elimina




Figura 16 - Gestione profili

Il pulsante **Utenti Locali**, invece, invoca una maschera Gestione Utenti (Figura 17) costituita da:

- un filtro, per la ricerca degli utenti;
- un report, per la visualizzazione dei risultati della ricerca;
- un form, per la modifica dei dati associati all'utente (Cognome, Nome, Login, Attiva);
- un pulsante *Associa profili*, cliccando il quale si ha accesso ad una nuova maschera per l'associazione di un determinato profilo all'utente selezionato (Figura 18).

Associa Profilo Utente

Codice	Profilo
1	AMMINISTRATORE



Codice

1

Profilo

AMMINISTRATORE

AMMINISTRATORE

TEST




+

Ricerca Utenti

Associa Profili

Gestione Utenti

Codice	Cognome	Nome	Attivo
0	AMMINISTRATORE	DI SISTEMA	S
1	Pallino	Pinco	S



First

Previous

1

Next

Last

Codice

0

Cognome*

AMMINISTRATORE

Nome*

DI SISTEMA

Login*

SUPER

Attiva*

☒ SI

☐ NO

Clear

Salva 




Elimina

Figura 18

Figura 17 – Maschera Gestione Utenti

Associa Profilo Utente

Codice	Profilo
1	AMMINISTRATORE



Codice

1

Profilo

AMMINISTRATORE

AMMINISTRATORE

TEST

Figura 18 - Maschera Associa Profilo Utente

5. Oggetti astratti

Di seguito si andranno a descrivere nel dettaglio tutti gli oggetti del BMF3 e la relativa procedura di configurazione, ricordando che sia la funzionalità *“Gestione Oggetti”* sia quella *“Configura Oggetti”* richiedono la compilazione di:

- Parametri di configurazione;
- Json file;
- campo Query.

Il BMF3 mette a disposizione una vasta gamma di oggetti, ciascuno corrispondente ad una specifica funzionalità, con i quali si “compongono” le maschere della web-app.

Tali funzionalità comprendono, ad esempio, report in forma tabellare o grafica, filtri di ricerca, form di inserimento, pulsanti di navigazione.

5.1 Oggetti per la visualizzazione

Partiamo con il considerare gli oggetti di visualizzazione dei dati.

5.1.1 Report

L'oggetto di tipo *Report* viene utilizzato per visualizzare i dati in formato tabellare. Ogni *report* supporta la paginazione.

Step 1 - Parametri di configurazione

ID*: identificativo numerico dell'oggetto, settato automaticamente dal BMF.

Nome*: nome dell'oggetto; per convenzione, il nome degli oggetti *report* inizia con “V_” (sono, di fatto, delle *viste* di dati).

Tipo*: *report*;

Descrizione: descrizione dell'oggetto;

DB: pool su cui verrà eseguita la query; se non specificato, sarà utilizzato il DB di default del framework.

Per gli oggetti di tipo *report*, i campi relativi ai parametri Link, LinkParams, Ordine non hanno significato. I campi contrassegnati con l'asterisco sono quelli obbligatori.

Step 2 – Json

Come vedremo, è soprattutto la configurazione del campo Json a differire tra un oggetto BMF e l'altro.

Partiamo da un esempio: la configurazione Json relativa al *report* in Figura 19. La parte evidenziata in blu rappresenta la struttura comune a tutti gli oggetti BMF, dove l'attributo *type* indica che si sta configurando un *report*; l'attributo *report*, invece, contiene tutte le proprietà di configurazione caratteristiche del relativo oggetto e definisce la struttura della tabella:

```
{"type": "report",
"report": {
  "title": "Esempio Report Base",
  "collapse": "N",
  "customization": "",
  "cols": { "RIC.T_RICOVERO_ID": { "label": "Id Ricovero" },
            "RIC.T_PAZIENTE_ID": { "label": "Codice Paziente" },
            "REP.T_REPARTO_NOME": { "label": "Reparto" },
            "T_RICOVERO_DATA_INIZIO": { "label": "Data Inizio" },
            "T_RICOVERO_DATA_FINE": { "label": "Data Fine" } },
  "idxs": ["RIC.T_RICOVERO_ID", "RIC.T_PAZIENTE_ID", "REP.T_REPARTO_NOME",
            "T_RICOVERO_DATA_INIZIO", "T_RICOVERO_DATA_FINE"]}}
```

Esempio Report Base				
Id Ricovero	Codice Paziente	Reparto	Data Inizio	Data Fine
1	1	Cardiologia pediatrica	01/01/2018	05/01/2018
2	1	Cardiologia pediatrica	01/01/2017	10/01/2017
3	2	Diagnostica ematochimica	01/01/2018	05/01/2018
4	3	Nefrologia	01/01/2018	05/01/2018
5	3	Nefrologia	01/01/2017	10/01/2017
6	4	Chirurgia pediatrica	01/01/2018	05/01/2018
7	2	Nefrologia	01/01/2018	05/01/2018

Figura 19 - Esempio oggetto Report

Dunque, editando il campo Json, sarà possibile specificare le seguenti proprietà:

- **title**: un titolo da assegnare al report;

- **collapse** (S/N): attributo che permette di “chiudere” il *report* ovvero di nascondere in caso di selezione di un record (utile nel caso in cui il report sia associato ad un *input-form* per data entry);
- **display**: se non definito, la visualizzazione sarà quella tradizionale mentre, se valorizzato con *column*, verrà utilizzata la visualizzazione per colonna;
- **customization**: il nome del modello personalizzato, implementato lato client, da applicare all'oggetto *report* che stiamo creando (per maggiori dettagli sulla gestione delle personalizzazioni si rimanda allo specifico capitolo);
- **cols***: hash map contenente tutti i campi che si vogliono gestire nel *report*. La chiave del hash è data dal nome del campo e per ognuno di essi si potranno definire gli attributi:

- **label**: etichetta della colonna (di default viene usato il nome del campo),
- **orderBy** (S/N): per attivare l'opzione di ordinamento su quella colonna;

- **idxs***: vettore contenente l'elenco dei nomi dei campi da visualizzare, in ordine, nella tabella. I nomi indicati sono i nomi dei vari campi che compongono il risultato della query. Se tale vettore non viene specificato verrà inizializzato posizionando i campi nell'ordine in cui si trovano nella query.

L'asterisco indica le proprietà obbligatorie.

Step 3 - Query

Nel campo Query viene scritta l'istruzione SQL per il caricamento dei dati da DB ovvero la query per il popolamento della tabella, la cui struttura è stata dichiarata nel campo Json; il nome dei campi restituiti dalla query deve essere coerente con quello delle chiavi dell'hash map definito nel Json.

In riferimento all'esempio di Figura 19, la query sarà:

```
SELECT RIC.T_RICOVERO_ID, RIC.T_PAZIENTE_ID, REP.T_REPARTO_NOME,  
TO_CHAR(RIC.T_RICOVERO_DATA_INIZIO, 'DD-MM-YYYY') AS  
T_RICOVERO_DATA_INIZIO,  
TO_CHAR(RIC.T_RICOVERO_DATA_FINE, 'DD-MM-YYYY') AS T_RICOVERO_DATA_FINE,  
FROM T_RICOVERO RIC LEFT JOIN T_REPARTO REP ON  
RIC.T_REPARTO_ID=REP.T_REPARTO_ID
```

Come è possibile vedere in Figura 19, il template di ogni report contiene di default tre pulsanti, rispettivamente per:

- creare il file PDF contenente la tabella che si sta visualizzando;
- esportare la tabella e i dati in essa contenuti in un file .excel;
- stampare la tabella.

Vediamo ora un esempio relativo ad una visualizzazione tabellare più complessa, capace di presentare informazioni di sintesi calcolate in maniera automatica sui dati di interesse. Supponiamo di voler dare una rappresentazione sintetica dei ricoveri, nella quale, per ogni sede, siano visualizzati il numero di ricoveri e la relativa percentuale rispetto al totale dei ricoveri su tutte le sedi (Figura 20).

In questo caso, il campo Json sarà:

```
{"type": "report",  
"report": {  
  "title": "Trend Ricoveri per reparto",  
  "cols": { "REPARTO": { "label": "Reparto"},  
            "QUANTITA": { "label": "Quantità" },  
            "QUANTITA_PERC": { "label": "%" }  
        },  
  "totals": { "QUANTITA": { "total": 0 } },  
  "calcs": { "QUANTITA": [ { "col": "QUANTITA_PERC", "type": "perc" } ] },  
  "idxs": [ "REPARTO", "QUANTITA", "QUANTITA_PERC" ]  
}
```

Trend Ricoveri per reparto		
Reparto	Quantità	%
Cardiologia pediatrica	2	28.57%
Chirurgia pediatrica	1	14.29%
Diagnostica ematochimica	1	14.29%
Nefrologia	3	42.86%
TOTALE	7.00	
		

Figura 20 - Esempio Report visualizzazione complessa

Nella configurazione Json, sono stati utilizzati due nuovi attributi:

- **totals**: hash map contenente tutti i campi di cui si vuole calcolare il totale (anche in questo caso, come per la proprietà *cols*, la chiave del hash è data dal nome del campo); il totale, come mostrato in Figura 20, sarà visualizzato alla fine della relativa colonna.

- **calcs**: hash map contenente tutti i campi dei quali si vuole calcolare il valore percentuale; nell'esempio considerato, è stato scelto di calcolare i valori percentuali del campo QUANTITA: tali valori saranno visualizzati in una nuova colonna del report, creata ad hoc (nel nostro caso QUANTITA_PERC). Dunque, ancora una volta la chiave del hash è data dal nome del campo su cui si vuole attuare il calcolo, mentre il valore è dato da un array di attributi:

- **col**: indica il nome della colonna da costruire ad hoc e destinare alla visualizzazione dei valori percentuali,
- **type**: indica la formula da applicare ("perc").

Dunque, il relativo codice SQL da scrivere nel campo Query sarà:

```
SELECT T_RICOVERO.T_RICOVERO AS SEDE,  
COUNT(T_RICOVERO.T_RICOVERO_ID) AS QUANTITA  
FROM T_RICOVERO  
GROUP BY T_RICOVERO.T_RICOVERO_SEDE  
ORDER BY T_RICOVERO.T_RICOVERO_SEDE
```

Le opzioni *totals* e *calcs* sono valide ovviamente soltanto se applicate a colonne di tipo numerico.

5.1.2 Approfondimento - Visualizzazione per colonna

È possibile scegliere di organizzare i record rappresentati in un oggetto *report* secondo una visualizzazione per colonna, semplicemente agendo in fase di configurazione del relativo Json file ed assegnando all'attributo *display* il valore *column*; ad esempio:

```
{ "type": "report",  
  "report": {  
    "title": "Dizionario Edifici",  
    "cols": { "T_EDIFICIO_CODICE": { "label": "Id" },  
              "T_EDIFICIO_DESC": { "label": "Codice" } },
```

```
"idxs": ["T_EDIFICIO_CODICE", "T_EDIFICIO_DESC"],  
"display": "column"  
}}
```

Questa opzione di visualizzazione non è compatibile con le opzioni *totals* e *calcs*, precedentemente illustrate (pag. 35), le quali richiedono necessariamente una visualizzazione standard per riga.




Dizionario Edifici	
Id	1
Codice	POLO A
Id	2
Codice	POLO B
Id	3
Codice	POLO C
<div></div>	

Figura 21 - Report con template di visualizzazione per colonna

5.1.3 Chart

L'oggetto di tipo Chart viene utilizzato per visualizzare i dati in un formato diverso da quello classico tabellare: il chart, infatti, è un grafico, per la creazione del quale il BMF3 ricorre alla libreria [Google Charts](#). I grafici disponibili sono:

- Pie: grafico a torta,
- Donut: grafico a ciambella,
- Bar: grafico a barre orizzontali,
- Column: grafico a barre verticali,
- Histo: grafico a istogramma,
- Line: grafico a linee,
- Area: grafico ad aree,
- Geo: grafico a mappa,
- Tachi: tachimetro,
- Gant: diagramma di Gantt,
- BoxPlot: grafico di tipo box plot,
- Candle: grafico di tipo candlestick.

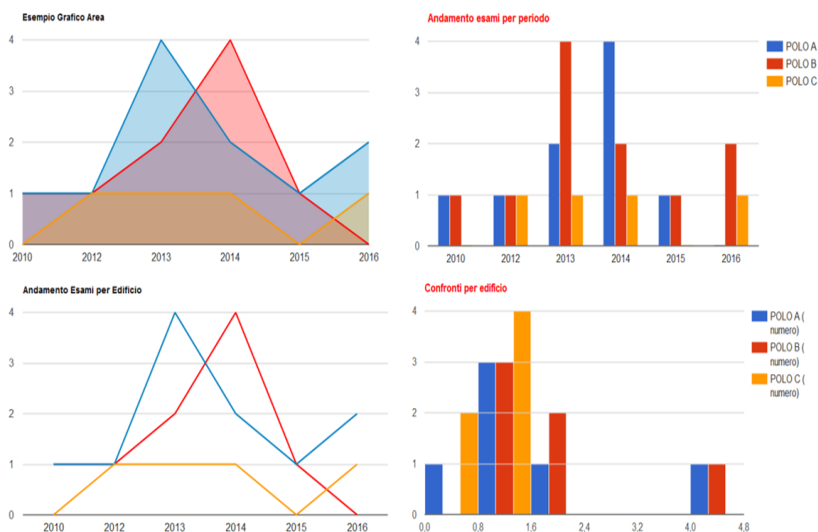


Figura 22 - Esempi di oggetti Chart

Step 1 – Parametri di configurazione

Per ognuna delle tipologie sopra elencate, la modalità di configurazione dei parametri non cambia:

ID*: identificativo numerico dell'oggetto, settato automaticamente dal BMF.

Nome*: nome del grafico; per convenzione, il nome degli oggetti di tipo *chart* inizia con "G_";

Tipo*: *chart*;

Descrizione: descrizione dell'oggetto;

DB: se non specificato, il DB di default del framework.

I campi relativi ai parametri Link, LinkParams, Ordine non hanno significato per gli oggetti *chart*. I campi contrassegnati con l'asterisco sono quelli obbligatori.

Step 2 – Json

Si riporta di seguito un esempio di configurazione per un grafico di tipo *column*, ma tale configurazione risulta valida anche per le altre tipologie.

La parte evidenziata in blu rappresenta la struttura comune a tutti gli oggetti: l'attributo *type* indica che si sta configurando un grafico mentre l'attributo *chart* contiene tutte le opzioni di configurazione proprie di questo particolare oggetto.

```
{
  "type": "chart",
  "chart": {
    "title": {"label": "Trend Ricoveri Ultimo Triennio per Sede",
              "fontSize": "14", "color": "Red"},
    "type": "Column",
    "height": "60",
    "width": "60",
    "fontSize": "12",
    "legend": "N",
    "seriesColors": ["#61AE24", "#D70060"],
    "xaxis": {"label": "Sede", "field": "SEDE"},
    "series": [ {"label": "ANNO", "field": "DIMESSI", "dynamic": "S"} ] } }
```

Tali proprietà comprendono:

- **type***: indica il tipo di grafico da realizzare (Bar, Pie, Line, Area, Column, Histo, Donut, Geo, Tachi, Gant, BoxPlot, Candle);

- **title**: oggetto per l'impostazione del titolo del grafico, attraverso la valorizzazione di tre attributi:

label: testo visualizzato come titolo del grafico;

fontSize: dimensione del font da assegnare al testo;

color: colore da assegnare al testo;

- **width** : larghezza del grafico;

- **height**: altezza del grafico;

- **fontSize**: dimensione del font da assegnare agli assi x e y e alla legenda;

- **legend** (S/N, S valore di default): opzione di visualizzazione della legenda;

- **seriesColors**: array di colori da utilizzare per le serie (di default saranno impostati quelli scelti come standard dalla libreria Google);

- **axis***: oggetto che definisce le labels (i gruppi, ovvero le variabili indipendenti) del grafico attraverso gli attributi:

- **label***: etichetta,

- **field***: campo della query da usare per popolare tale gruppo del grafico;

- **series***: oggetto che definisce le possibili n-series (ovvero le variabili dipendenti, ciò che per ogni gruppo si vuole rappresentare), attraverso:

- **label***: etichetta della serie;

- **field***: valore della serie;

- **dynamic** (S/N): attributo che indica se la label è fissa oppure è dinamica (ossia ottenuta come valore risultante dalla query SQL).

- **geo***: oggetto contiene le proprietà obbligatorie nel caso di configurazione di un *Geo Chart*, e richiede solo per tale tipologia di grafico:

region*: indica la regione che deve essere rappresentata nel grafico (IT per Italia, 150 per Europa, etc. Per maggiori dettagli sui possibili valori da assegnare a tale attributo si faccia riferimento alla documentazione ufficiale dei Geo Google Charts);

resolution*: risoluzione dei confini (countries, provinces, metros);

colors: scala di colori;

displayMode: tipo di rappresentazione (regions, markers).

L'asterisco indica le proprietà obbligatorie.

Step 3 – Query

Il codice SQL recupera dal DB i dati che si vogliono rappresentare. È importante che vi sia coerenza tra il nome dei campi restituiti da questa select e quelli utilizzati nella configurazione degli attributi *xaxis* e *series* nel Json.

Vediamo ora nel dettaglio qualche esempio.

- *Pie e Donut per la rappresentazione dei trend di ricovero*

Le sedi dei ricoveri rappresentano la variabile indipendente (*xaxis*); per ogni sede, si vuole rappresentare la relativa quantità di ricoveri avvenuti (variabile dipendente, *series*).

Json

```
{
  "type": "chart",
  "chart": {
    "title": {"label": "Ricoveri per sede"},
    "type": "Pie",
    "xaxis": {"label": "Sede", "field": "SEDE"},
    "series": [ {"label": "QUANTITA", "field": "QUANTITA"} ],
  }
}
```

Scegliendo per l'attributo type il valore *Pie* si ottiene il grafico a torta piena (Figura 23 - a) mentre scegliendo il valore *Donut* si ottiene il grafico a torta vuota (Figura 23 - b).

Query

```
Select    R.T_RICOVERO_SEDE AS SEDE,
          COUNT(R.T_RICOVERO_ID) AS QUANTITA
from      T_RICOVERO R
group by  R.T_RICOVERO_SEDE
```

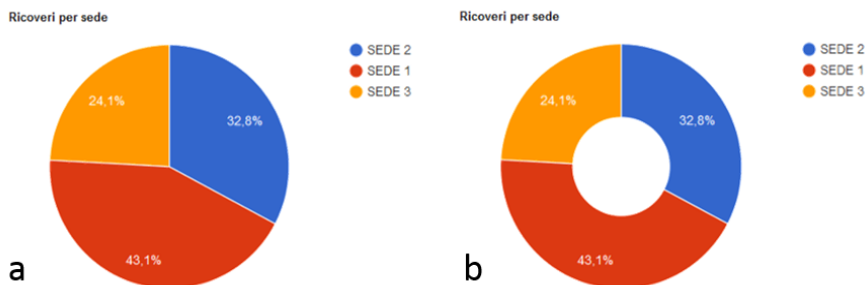


Figura 23 - Esempio Pie e Donut Chart

Come si osserva dall'esempio, vi è coerenza tra il nome dei campi restituiti dalla query e quelli utilizzati nella configurazione degli attributi *xaxis* e *series* nel Json.

- Bar per la rappresentazione del trend degli esami, suddivisi per tipologia.

Per ogni tipologia di esame (variabile indipendente *xaxis*) se ne vuole rappresentare la quantità effettuata ogni anno (la variabile dipendente *series* sarà quindi QUANTITA, valore restituito da una select che raggruppa i dati per tipologia di esame e per anno). Dunque, nel grafico ottenuto ogni barra colorata si riferisce ad una specifica annualità (Figura 24).

Json

```
{
  "type": "chart",
  "chart": {
    "type": "Bar",
    "title": {
      "label": "Trend Esami per Tipo", "color": "#000000",
    },
    "width": "1000",
    "xaxis": {
      {"label": "Esame", "field": "ESAME"},
    },
    "series": {
      [{"label": "ANNO", "field": "QUANTITA", "dynamic": "S"}],
    },
    "seriesColors": ["#ee0b2d"]
  }
}
```

Query

```
Select    T_ESAME_DESCR AS ESAME,
          COUNT(T_ESAME_OID) AS QUANTITA,
          TO_CHAR(EXTRACT(YEAR FROM T_ESAME_DATA)) AS ANNO
from      T_ESAME
group by  EXTRACT(YEAR FROM T_ESAME_DATA), T_ESAME_DESCR
order by  EXTRACT(YEAR FROM T_ESAME_DATA)
```

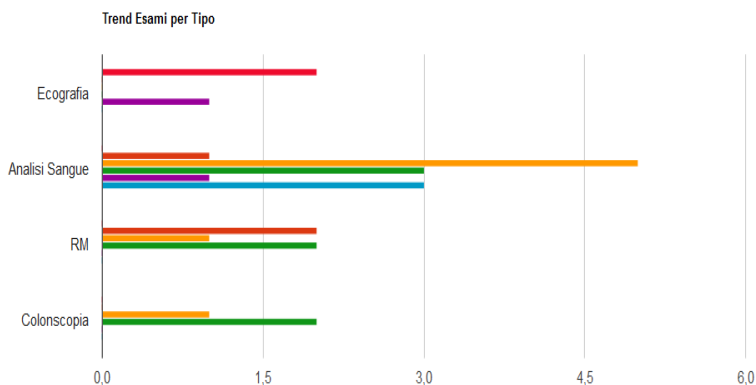


Figura 24 - Esempio Bar Chart

- Geo Chart per la rappresentazione di distribuzione su area geografica.

In questo caso, i gruppi (variabile indipendente) sono le regioni italiane (Figura 25). I valori di COUNTRY, campo restituito dalla query che recupera i dati da DB, saranno i nomi delle regioni (in lingua italiana).

Json

```
{"type": "chart",
"chart": {
  "type": "Geo",
  "title":
    {"label": "Esempio Geo Chart", "color": "#000000"},
  "geo": {
    "region": "IT",
```

```

    "resolution": "provinces",
    "colors": ["orange", "red"]},
    "width": "1000",
    "xaxis": {"label": "COUNTRY", "field": "COUNTRY"},
    "series": [{"label": "POPULARITY", "field": "QTA"}]

```

}}

Query

select COUNTRY, QTA from V_GEO

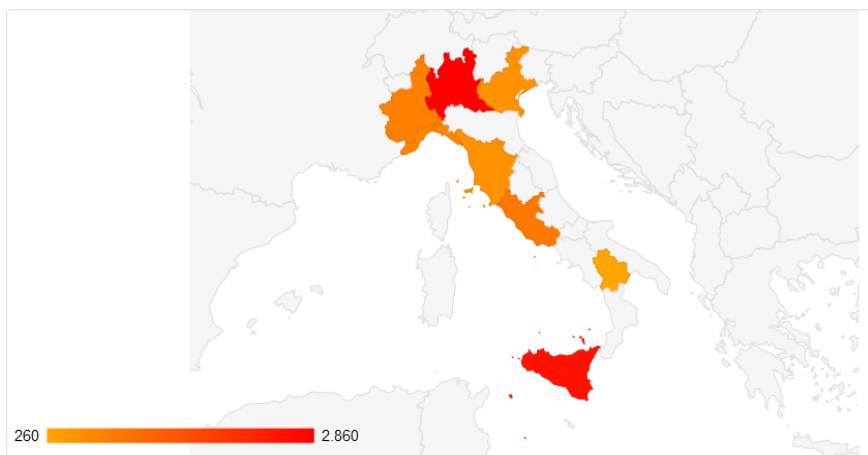


Figura 25 - Esempio Geo Chart

- Grafici annidati

I grafici annidati sono due grafici collegati tra loro in modo tale che il primo agisca da filtro per i dati visualizzati nel secondo. Per configurare questa funzionalità è necessario aggiungere nel file Json del primo Chart, quello filtrante, la seguente porzione di configurazione:

```

"subreport": {
    "querystring": "idObject=G_TrendRicoveriSesso",
    "breadcrumb": "REPARTO",
    "params": [
        {"value": "T_REPARTO_ID", "name": "T_REPARTO_ID",
        "session": "N", "operator": "equal"}]}

```

Le proprietà dell'oggetto subreport sono (quelle indicate con l'asterisco sono obbligatorie):

- **querystring***: stringa da passare alla chiamata del dispatcher indicante gli oggetti da caricare;
- **breadcrumb**: nome del campo, restituito della query che genera il grafico, da usare per valorizzare il *breadcrumb* (se non valorizzato il *breadcrumb* non verrà generato);
- **params***: array di oggetti contenente l'elenco dei campi che agiranno da filtro, attraverso la definizione degli attributi:
 - **name***: nome del parametro;
 - **value***: nome del campo SQL da cui recuperare il valore da assegnare al parametro;
 - **session**: attributo che indica se il parametro deve essere gestito in sessione o usato per comporre una condizione SQL (S/N);
 - **operator**: tipo di operatore da utilizzare nella condizione SQL.

Nell'esempio considerato, il primo grafico mostra la distribuzione dei ricoveri nei diversi reparti della struttura ospedaliera. Tale grafico agisce come filtro per la visualizzazione dei dati nel chart annidato, il quale differenzia i ricoveri per sesso, considerando il reparto selezionato nel primo grafico.

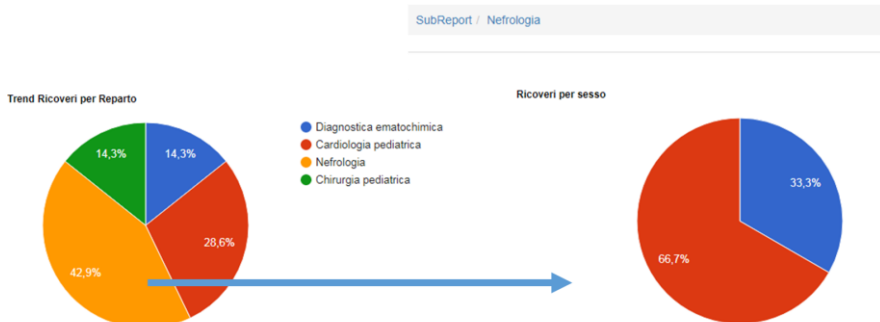


Figura 26 – Esempio grafici annidati

5.1.4 Calendar

Questo tipo di oggetto è un particolare report per la rappresentazione dei dati all'interno di un calendario mensile.

Step 1 - Parametri di configurazione

ID*: identificativo numerico dell'oggetto, settato automaticamente dal BMF;

Nome*: nome dell'oggetto; per convenzione, il nome degli oggetti *calendar* inizia con "V_";

Tipo*: *calendar*;

Descrizione: descrizione dell'oggetto;

DB: pool su cui verrà eseguita la query; se non specificato, il DB di default del framework.

I campi relativi ai parametri Link, LinkParams, Ordine non hanno significato per gli oggetti di tipo *calendar*. I campi contrassegnati con l'asterisco sono quelli obbligatori.

Step 2 – Json

La configurazione del campo Json è molto semplice: sarà necessario specificare soltanto l'attributo *title*, titolo del report; ad esempio:

```
{"type": "calendar",  
"calendar": {  
    "title": "Esempio Calendario"}}
```

Step 3– Query

L'aspetto fondamentale per il buon funzionamento di questo tipo di report è la costruzione della query da eseguire, la quale dovrà rispettare il seguente formato:

```
SELECT  
to_char (DATA, 'DD') as day_of_month, ORA AS time, NOMINATIVO AS note  
FROM (  
    select ... as DATA, ... as ORA, ... as NOMINATIVO  
    from ...  
    where ...  
    and (TRUNC (... , 'MM') = to_date ('<bmf_day>', 'MM/YYYY'))
```

Il primo campo indica il giorno del mese che si sta rappresentando (**day_of_month**), gli altri due rappresentano le informazioni che si vogliono visualizzare all'interno delle celle del calendario (ad esempio ora e nota).
Nella clausola where è necessario specificare il mese da considerare, settato nel parametro <bmf_day>.

Esempio Calendario

← Mese Precedente

OTTOBRE 2018

Mese Successivo →

Lun	Mar	Mer	Gio	Ven	Sab	Dom
1	2	3	4	5	6	7
8	9	10	11 Chiamata SP numero 1	12	13	14
15	16	17	18 Chiamata SP numero 2	19	20	21
22	23	24	25	26	27	28
29	30	31				

Figura 27 - Esempio oggetto Calendar

5.2 Oggetti per il data entry

L'oggetto BMF per l'operazione di data entry è l'*input-form*.

5.2.1 Input-form

L'oggetto di tipo *input-form* permette di fare data entry sulle tabelle del DB; permette inoltre l'inserimento di dati da parte dell'utente per l'attivazione di un'azione lato server side.

Step 1 - Parametri di configurazione

ID*: identificativo numerico dell'oggetto, settato automaticamente dal BMF;

Nome*: nome dell'oggetto (per convenzione, il nome degli oggetti *input-form* inizia con "T_");

Tipo*: *inputForm*;

Descrizione: descrizione dell'oggetto;

DB: pool su cui verrà eseguita la query (se non specificato, il DB di default del framework).

I campi relativi ai parametri Link, LinkParams, Ordine non hanno significato per gli oggetti di tipo *input-form*. I campi contrassegnati con l'asterisco sono quelli obbligatori.

Step 2 – Json

Partiamo da un esempio: la configurazione Json relativa all'*input-form* di Figura 28. La parte evidenziata in blu rappresenta la struttura comune a tutti gli oggetti: l'attributo *type* indica che si sta configurando un form di inserimento dati mentre l'attributo *inputForm* contiene le opzioni di configurazione proprie del relativo oggetto.

```
{  
  "type": "inputForm",  
  "inputForm": {  
    "title": "Inserisci nuovo paziente",  
    "tableName": "T_PAZIENTE",
```



```
"toUpperCase": "S",
"fields":[
  {
    "id":"T_PAZIENTE_ID",
    "name":"T_PAZIENTE_ID",
    "label":"ID",
    "enable":"N",
    "session":"N",
    "type":"text"
  },
  {
    "id":"T_PAZIENTE_COGNOME",
    "name":"T_PAZIENTE_COGNOME",
    "label":"Cognome",
    "session":"N",
    "type":"text",
    "validation": {"required": "true"}
  },
  {
    "id":"T_PAZIENTE_NOME",
    "name":"T_PAZIENTE_NOME",
    "label":"Nome",
    "session":"N",
    "type":"text",
    "validation": {"required": "true"}
  },
  {
    "id":"T_PAZIENTE_CF",
    "name":"T_PAZIENTE_CF",
    "label":"CF",
    "session":"N",
    "type":"text",
    "validation": {"required": "true"}
  },
  {
```

```
"id": "T_PAZIENTE_DATA_NASCITA",
"name": "T_PAZIENTE_DATA_NASCITA",
"label": "Data di Nascita",
"session": "N",
"type": "text",
"typeObj": "Date",
"validation": {"required": "true"}
},
{
  "id": "T_PAZIENTE_SESSO",
  "name": "T_PAZIENTE_SESSO",
  "label": "Sesso",
  "session": "N",
  "type": "radio",
  "list": [{"val": "M", "descr": "M"}, {"val": "F", "descr": "F"}],
  "validation": {"required": "true"}
}],
"customization": "",
"pk": {"auto": "S", "fields": ["T_PAZIENTE_ID"]},
"buttons": [{"id": "btn_save", "name": "btn_save", "label": "Salva",
"iconClass": "glyphicon glyphicon-floppy-saved"}
}]
```

Inserisci nuovo paziente

The form consists of the following elements:

- ID**: A text input field with a light gray background.
- Cognome**: A text input field.
- Nome**: A text input field.
- CF**: A text input field.
- Data di Nascita**: A date picker field with a calendar icon on the right.
- Sesso**: Two radio buttons labeled **M** and **F**.
- Buttons**: Two blue buttons at the bottom right, labeled **Clear** and **Salva** (with a save icon).

Figura 28 - Esempio input-form

Dunque, editando il campo Json, è possibile definire le seguenti proprietà:

- **title**: titolo da assegnare all'*input-form*;
- **tableName***: nome della tabella del DB su cui si vuole fare data entry;
- **toUpperCase** (S/N): attributo utilizzato per convertire in maiuscolo i valori inseriti nei *fields* del *form*;
- **pk***: attributo per la composizione della *primary key*. È possibile gestire PK automatiche, composte da un unico campo numerico che si auto-incrementa, oppure PK (singole o composte) la cui valorizzazione è demandata all'utente. L'attributo pk è un array composto dagli attributi:

- **auto** (S/N): se il suo valore è S, la chiave viene auto-generata lato server e dovrà essere singola, ovvero composta da un unico campo numerico;
- **fields**: nome dei campi del form che andranno a comporre la PK.

- **fields***: oggetto per la definizione dei campi che andranno a comporre l'*input-form*. Per ciascun campo sarà possibile specificare i seguenti attributi:

- 1) **id***: identificativo univoco (deve coincidere con il nome del campo della tabella del DB su cui si intende fare data entry);
- 2) **name***: nome SQL del campo;
- 3) **label**: etichetta descrittiva del campo (se non valorizzato, viene usato il valore scelto per l'attributo *name*);

4) **type***: tipo del campo; questa proprietà può assumere i seguenti valori:

- **select** (il valore del campo sarà selezionabile dall'utente tra quelli presenti all'interno di una lista),
- **hidden** (il campo è nascosto, non viene visualizzato dall'utente nella maschera),
- **text** (il valore del campo sarà definito da parte dell'utente mediante digitazione da tastiera - testo libero);
- **radio** (l'utente può selezionare il valore del campo scegliendo tra quelli presenti all'interno di un elenco di radio-button – selezione mutuamente esclusiva);
- **checkbox** (l'utente può selezionare il valore del campo scegliendo tra quelli presenti all'interno di un elenco check-list – selezione non mutuamente esclusiva).

Tutte queste diverse tipologie saranno approfondite nei paragrafi successivi.

5) **session**: indica se il parametro deve essere inserito tra quelli in sessione;

6) **default**: indica il valore da attribuire automaticamente al campo nel caso in cui, durante la compilazione del *form*, questo non fosse valorizzato;

7) **validation**: attributo attraverso cui impostare i controlli di validazione, ad esempio nel caso in cui debba essere obbligatoriamente inserito un valore nel campo, oppure per controllare che il valore inserito sia maggiore/minore di una certa quantità, o che rispetti un certo pattern di validazione (mail, telefono...) ecc. L'utilizzo di questo attributo sarà approfondito nel paragrafo 5.2.5;

6) **placeholder**: è il valore visualizzato all'interno del campo ancora vuoto.

- **customization**: nome del modello custom, implementato lato client, da applicare al form di inserimento, ad esempio per personalizzare gli eventi legati ai pulsanti; il modello custom è un'estensione js del modello base *inputForm*. Per maggiori dettagli sulla gestione delle personalizzazioni si rimanda allo specifico capitolo (6).

- **buttons**: elenco dei pulsanti che compaiono nella maschera, in coda all'*input-form*; oltre ai pulsanti di default (Save, Delete, Clear, Save As) è possibile aggiungerne altri attraverso la procedura di customizzazione. Le proprietà di ogni pulsante comprendono:

1) **id***: identificativo del pulsante (per quelli di default va mantenuto il nome originale);

- 2) **name***: nome del pulsante (per quelli di default va mantenuto il nome originale);
- 3) **label***: etichetta da associare al pulsante;
- 4) **iconClass**: eventuale icona da associare al pulsante;
- 5) **enabled** (S/N): abilitazione del pulsante; se il valore è N, il pulsante è disabilitato.

Step 3 - Query

Nel campo query viene definita la select per il caricamento dei dati da DB e il loro posizionamento nei vari elementi dell'*input-form*; è importante sottolineare la necessità di corrispondenza tra il valore dell'attributo *id* scelto per tutti i fields configurati nel file Json e il nome dei campi restituiti dalla query e riferiti alle colonne della tabella del DB su cui si desidera fare data entry.

```
select T_ANAG_ID,  
T_ANAG_COGNOME,  
T_ANAG_NOME,  
T_ANAG_CF,  
TO_CHAR(T_ANAG_DATA_NASCITA,'DD-MM-YYYY') AS  
T_ANAG_DATA_NASCITA,  
T_ANAG_ETA,  
T_ANAG_SESSO  
FROM T_ANAG  
WHERE <condizioni>
```

5.2.2 Tipologie di text-field

Text

Nel campo di tipo *text* l'utente può inserire liberamente un valore tramite tastiera.

```
{id:"Nome",
name:"NOME",
label:"Nome",
session:"N",
default:"",
placeholder:"",
type:"text"}
```


Nome

Tizio

Nel caso in cui il campo sia destinato a contenere un valore di tipo **data**, è possibile impostare l'attributo *typeObj* in modo da sfruttare l'utility Calendario, che semplifica l'inserimento.

```
{id:"DATA",
name:"DATA",
label:"Data di Nascita",
session:"S",
default:"",
placeholder:"",
type:"text",
typeObj:"Date"}
```

Data di Nascita



July 2018

Su	Mo	Tu	We	Th	Fr	Sa
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

Select

L'utente sceglie il valore di un campo di tipo *select* tra quelli contenuti all'interno di un menu di selezione a tendina. L'elemento *select* può essere configurato in due modi diversi: definendo direttamente la lista dei dati con cui popolare il menu di selezione (caso 1), oppure recuperando quei dati come risultato di una query su DB (caso 2).

Caso 1) Definizione manuale della lista attraverso l'attributo *list*:

```
{'id': 'FA',
 'name': 'FA',
 'label': 'Fibrillazione
atriale',
 'session': 'N',
 'default': '',
 'placeholder': '',
  'type': 'select',
  'list': [   {'val': '1', 'descr': 'SI'},
              {'val': '0', 'descr': 'NO'}
]}
```

Fibrillazione atriale

----	▼
SI	
NO	

Caso 2) Popolazione dinamica della lista attraverso l'attributo *select*:

```
{'id': 'PAT',
 'name': 'PAT',
 'label': 'Tipo patologia',
 'session': 'N',
 'default': '',
 'placeholder': '',
  'type': 'select',
  'select': {
    'idObject': 'S_Patologia',
    'key': 'T_PATOLOGIA_ID',
    'value': 'T_PATOLOGIA_DESC'}}
```

Tipo patologia

----	▼
CARDIOMIOPATIA	
CANALOPATIA	
ANOMALIA CORONARICA	
MISCELLANEA	

La query per il recupero dei dati da DB è contenuta in uno specifico oggetto BMF, in questo esempio chiamato `S_Patologia`: si tratta di un oggetto `SELECT` (per maggiori dettagli si faccia riferimento al paragrafo 5.5.5); l'attributo `idObject` che richiama tale oggetto è obbligatorio mentre sono opzionali gli attributi `key` e `value`, i quali specificano quali campi della query scritta in `S_patologia` utilizzare rispettivamente come valore-chiave e come nome descrittivo visualizzato nel menu di selezione; se non specificati si presuppone che la select SQL sia stata scritta rispettando il seguente formalismo:

```
select xx as VAL, yy as DESCR from ...
```

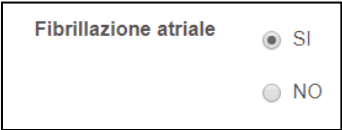
La selezione è mutuamente esclusiva. Per renderla non esclusiva è necessario attivare la proprietà *multiple* del *field Json* di tipo *select*:

```
...  
'type': 'select',  
'multiple': 'S'  
...
```

Radio Button e Checkbox

Analogamente all'elemento *select*, le tipologie *radio* e *check-box* possono essere configurate definendo direttamente la lista dei dati con cui popolare le opzioni di selezione, oppure recuperando i dati tramite query su DB. Nel caso di tipologia *radio*, la scelta tra le opzioni di selezione, rappresentate da una serie di *radio-button*, sarà mutuamente esclusiva mentre nel caso di *check-box* potranno essere selezionati più valori contemporaneamente.

Caso 1) `{'id': 'FA',`
 `'name': 'FA',`
 `'label': 'Fibrillazione atriale',`
 `'default': '',`
 `'placeholder': '',`
 `'type': 'radio',`
 `'list': [` `{'val': '1', 'descr': 'SI'},`
 `{'val': '0', 'descr': 'NO'}`
 `]}`



The image shows a rectangular box containing the text "Fibrillazione atriale" in a bold, dark font. To the right of this text are two radio buttons. The top radio button is selected (filled with a dark dot) and is followed by the text "SI". The bottom radio button is unselected (empty circle) and is followed by the text "NO".


```
Caso 2) {'id': 'PAT',
        'name': 'PAT',
        'label': 'Tipo patologia',
        'session': 'N',
        'placeholder': '',
        'type': 'checkbox',
        'select': {
            'idObject': 'S_Patologia',
            'key': 'T_PATOLOGIA_ID', 'value': 'T_PATOLOGIA_DESC'}}
```

Tipo patologia

☒ CARDIOMIOPATIA

☐ CANALOPATIA

☐ ANOMALIA CORONARICA

☐ MISCELLANEA

Hidden

L'elemento hidden è un elemento non visibile, che risulta utile nel caso di definizione di condizioni di default, per le quali non si vuole lasciare alcuna possibilità di inserimento/modifica all'utente.

```
{ "id": "Reparto_id",
  "type": "hidden",
  "name": "Reparto_id",
  "default": "10" }
```

Nell'esempio precedente, il campo di inserimento relativo al codice di reparto non viene visualizzato nella maschera: il codice è sempre uguale al suo valore di default (10) e non deve essere impostato dall'utente.

Come valore di default può essere utilizzato un parametro presente in sessione, mediante l'opzione *fromSession*:

```
{ "id": "T_ANAG_ID",
  "name": "T_ANAG_ID",
  "label": "ID Paziente",
  "type": "hidden",
  "fromSession": {
    "key": "P_ID" }}
```

In questo esempio, al *field* relativo al codice del paziente viene associato il valore del parametro P_ID contenuto in sessione.

5.2.3 Elementi separatori

Gli elementi *separator* all'interno di un *input-form* creano un box di testo che separa o mette in evidenza alcune sezioni del form. Essi si collocano fisicamente tra i campi dell'*input-form* e per crearli è sufficiente introdurli, insieme agli altri campi, nell'array *fields*; la loro configurazione è però più semplice e richiede che siano specificati soltanto gli attributi *text* e *type*, come mostrato nel seguente esempio:

```
{
  type: 'separator'
  text: 'FORM DI INSERIMENTO DATI',
}
```

L'attributo *type* indica che stiamo introducendo un elemento separatore mentre l'attributo *text* contiene il testo che dovrà essere visualizzato all'interno della box.

The screenshot shows a web form with a light blue header bar containing the text "FORM INSERIMENTO DATI". Below the header, there are five input fields arranged vertically. The first field is labeled "ID" and is a simple text input. The second field is labeled "Codice" and is also a simple text input. The third field is labeled "Data Inizio" and has a calendar icon on its right side. The fourth field is labeled "Data Fine" and also has a calendar icon on its right side. The fifth field is labeled "Descrizione" and is a larger text input. Below the input fields, there are four buttons: "Clear", "Salva" (with a save icon), "Elimina", and "Save As".

Figura 29 - Esempio elemento Separator

Quella appena presentata è la configurazione di default; oltre a questa, possono essere definite classi alternative per personalizzare lo stile del componente separatore. Si riporta di seguito un esempio di regole di composizione per la creazione di un elemento separatore custom:

```
{ "text": "Associa medico ai Reparti",
  "type": "separator",
  "class": "info-big" }
```

dove *info-big* è il nome di una classe CSS contenuta nel foglio di stile del sito che stiamo sviluppando:

```
.info-big{
text-align: left;
font-size:18px;
font-weight: bold;
background-color:#D9A3CA; }
```

Associa medico ai Reparti

Figura 30 - Esempio separatore customizzato

5.2.4 Pulsante info su un campo

Il componente *info* costituisce una sorta di *tooltip* utile per fornire informazioni descrittive relativamente ad uno specifico campo di un *input-form*. Per utilizzare questa funzionalità, sarà sufficiente introdurre la proprietà *info* tra gli attributi del *field* in questione, assegnandole come valore la stringa che dovrà comparire all'interno del *tooltip*, come mostrato nell'esempio seguente:

```
{ 'id': 'T_MED_REP_UTENTE',
  'name': 'T_MED_REP_UTENTE',
  'label': 'Medico',
  'info': 'Medico della struttura',
  'type': 'select',
  'select': { 'idObject': 'S_Med' } }
```



Figura 31 - Esempio pulsante info

5.2.5 Validazione dei campi

La validazione dei campi ha lo scopo di verificare la corretta compilazione del form da parte dell'utente, attraverso controlli aggiuntivi e specifici vincoli, implementati lato client. Per configurare le regole di validazione è sufficiente aggiungere la proprietà *validation* in corrispondenza del *text-field* che si vuole validare.

Esempio:

```
{
  "id": "T_EMAIL",
  "name": "T_EMAIL",
  "label": "Indirizzo e-mail",
  "session": "N",
  "type": "text",
  "validation": {"required": "true", "pattern": "email"}
}
```

La regola di validazione sarà una di quelle di seguito riportate, oppure una combinazione di esse:

- *required (true/false)*: specifica se l'attributo è obbligatorio o meno. Nel caso in cui il suo valore sia *true*, l'utility controlla che l'utente abbia effettivamente inserito un dato in corrispondenza di quel campo del form;
- *acceptance (true/false)*: convalida qualcosa che deve essere accettato, ad esempio i termini di uso;
- *min*: convalida che il valore inserito in corrispondenza del campo sia un numero maggiore o uguale al valore specificato;
- *max*: convalida che il valore inserito in corrispondenza del campo sia un numero minore o uguale al valore specificato;
- *range*: convalida che il valore inserito in corrispondenza del campo sia un numero uguale o compreso nel range specificato;
- *length*: convalida che il valore inserito in corrispondenza del campo sia una stringa di lunghezza uguale al valore specificato;
- *minLength*: convalida che il valore inserito in corrispondenza del campo sia una stringa di lunghezza maggiore o uguale al valore specificato;

- *minLength*: convalida che il valore inserito in corrispondenza del campo sia una stringa di lunghezza minore o uguale al valore specificato;
- *rangeLength*: convalida che il valore inserito in corrispondenza del campo sia una stringa di lunghezza compresa nel range specificato;
- *oneOf*: convalida che il valore inserito in corrispondenza del campo sia corrispondente (case sensitive) a uno degli elementi specificati nell'array;
- *equalTo*: convalida che il valore inserito in corrispondenza del campo sia uguale a quello dell'input con il nome indicato;
- *pattern*: convalida che il valore inserito segua il pattern specificato tra uno di quelli disponibili:
 - ✓ *number*: corrisponde a qualsiasi numero (esempio -100.000,00);
 - ✓ *email*: corrisponde a un indirizzo email valido (esempio mail@example.com);
 - ✓ *url*: corrisponde a qualsiasi url (esempio <http://www.example.com>);
 - ✓ *digits*: corrisponde a qualsiasi cifra (esempio 0-9).

Se si necessita di una logica di validazione personalizzata è possibile estendere le regole disponibili aggiungendo le proprie. Viene fornita inoltre la possibilità di effettuare l'override dei metodi esistenti.

```
_.extend (Backbone.Validation.validators, {  
  myValidator: function(value, attr, customValue, model) {  
    if(value !== customValue){  
      return 'Messaggio di errore';  
    }  
  },  
  required: function(value, attr, customValue, model) {  
    if(!value){  
      return 'Metodo required personalizzato';  
    }  
  },  
});
```

5.2.6 Campi correlati

E' possibile definire una correlazione tra due campi di un form di inserimento ovvero è possibile filtrare i dati di uno sulla base del valore inserito nell'altro.

Ad esempio, si consideri una maschera per l'inserimento dei dati anagrafici di un paziente, tra i quali ci siano anche le informazioni relative al luogo di nascita: scegliendo una determinata Provincia, sarebbe utile che l'applicativo fosse in grado di filtrare il campo per la scelta del Comune, caricando tra le opzioni solo quelli correlati alla provincia selezionata. Per attivare questa utility è sufficiente aggiungere, nel campo che fa da filtro, l'oggetto **filterOn**, che definisce la presenza di un legame con un altro campo e che contiene due attributi:

- **id***: id del campo di cui si vogliono filtrare i dati (deve essere coerente con il relativo attributo id specificato nel Json);
- **keyName***: nome del campo SQL che agisce da filtro.

Esempio:

```
{ "id": "T_DIZ_REGIONI_CODICE ",
  "name": "T_DIZ_REGIONI_CODICE",
  "label": "Regione",
  "default": "",
  "type": "select",
  "select": [{"idObject": "S_REGIONI"}],
  "filterOn":
    { "id": "T_DIZ_COMUNI_CODICE",
      "keyName": "T_DIZ_REGIONI_CODICE" },
}
```

```
{ "id": "T_DIZ_COMUNI_CODICE ",
  "name": "T_DIZ_COMUNI_CODICE ",
  "label": "Comune",
  "default": "",
  "type": "select",
  "select": [{"idObject": "S_COMUNI"}]
}
```

L'oggetto "S_REGIONI" è un oggetto SELECT, contenente la query che recupera dal DB l'elenco delle regioni italiane (per popolare il campo T_DIZ_REGIONI_CODICE dell'*input-form*), mentre "S_COMUNI" è un oggetto SELECT, contenente la query che recupera dal DB l'elenco dei comuni (per popolare il campo T_DIZ_COMUNI_CODICE dell'*input-form*). La regione selezionata agirà da filtro sui risultati della select "S_COMUNI" e, di conseguenza, nel campo T_DIZ_COMUNI_CODICE del form saranno visualizzati soltanto i comuni appartenenti regione selezionata.

5.2.7 Inserimento multiplo

Il BMF prevede la possibilità di gestire l'inserimento di più record all'interno di una tabella del DB con una sola operazione di compilazione di uno specifico oggetto *input-form*. La condizione affinché si possa sfruttare la funzionalità di inserimento multiplo è che i record da inserire abbiano tutti una radice comune, costituita da n campi, e una parte variabile, costituita da un solo attributo, valorizzato attraverso un *input field* di tipo *checkbox*. Tale funzionalità risulta particolarmente utile nel caso in cui si abbiano, all'interno dello schema del DB, due tabelle di cui una madre e l'altra in relazione 1:N con essa, o due tabelle in relazione molti a molti tra loro. Si consideri proprio quest'ultimo caso: la tabella T_MED, contenente un certo numero di informazioni anagrafiche dei medici impiegati nella struttura ospedaliera, è in relazione N:M con T_REPARTO, tabella relativa ai reparti presenti nella struttura. Tale associazione prende forma nella tabella REL_MED_REP, avente come campi le due foreign key T_MED_ID e T_REPARTO_ID. L'esempio riportato di seguito si riferisce proprio al form per il data entry nella tabella REL_MED_ID, in cui è possibile notare come la funzionalità di inserimento multiplo possa essere attivata semplicemente scegliendo per l'attributo *type* dell'*input-form* il valore *multi*, e definendo in corrispondenza del campo multiplo gli attributi *multi* e *multiple* uguali a S:

```
{ "type": "inputForm",  
  "inputForm": {  
    "title": "",  
    "tableName": "REL_MED_REP",  
    "type": "multi",  
    "pk": { "auto": "N", "fields": [ "T_MED_ID", "T_REPARTO_ID" ] },
```

```
"fields": [
  {
    "text": "Associa medico ai Reparti",
    "type": "separator",
    "class": "info-big",
    {
      "id": "T_MED_ID",
      "name": "T_MED_ID",
      "label": "Medico",
      "info": "Medico della struttura",
      "type": "select",
      "select": {"idObject": "S_MED"}},
      {
        "id": "T_REPARTO_ID",
        "name": "T_REPARTO_ID",
        "label": "Reparto",
        "multi": "S",
        "type": "checkbox",
        "multiple": "S",
        "select": {"idObject": "S_REP"}}}
```

Medici		
	Cognome	Nome
Seleziona	VERDI	PLUTO
Seleziona	NERI	PINCO
Seleziona	BIANCHI	PIPPO
<div></div>		

Associa medico ai Reparti

Medico

BIANCHI PIPPO

?

Reparto

☐ Anestesia e rianimazione

☒ Cardiologia

☐ Diagnostica ematochimica

☒ Cardiologia pediatrica

☐ Chirurgia pediatrica

☐ Ostetricia e ginecologia

☐ Radiologia

☐ Nefrologia

☐ Psichiatria

Clear

Salva

Elimina

Figura 32 - Esempio inserimento multiplo

Tale configurazione, una volta selezionato il medico di interesse, permette di spuntare i reparti a cui lo stesso medico è associato. Dunque, T_MED_ID sarà la radice, fissa, dei record che saranno inseriti attraverso il form di inserimento nella tabella REL_MED_REP, mentre la parte variabile sarà il campo T_REPARTO_ID. Così come avviene per gli *input-form* standard, la *primary key* della tabella può essere gestita come un unico campo numerico che si auto incrementa oppure come uno o più campi valorizzati manualmente dall'utente. Mentre per il secondo scenario si mettono a disposizione le operazioni classiche di *insert/update/delete*, nel caso di chiavi gestite in maniera automatica è previsto soltanto l'*insert* dei dati.

5.2.8 Data entry con storicizzazione

La cancellazione logica dei dati viene gestita dal BMF3 attraverso l'attributo *Json archivia*. Tale funzionalità è nota anche come storicizzazione, in quanto permette di storicizzare le operazioni eseguite sui record attraverso il tracciamento delle seguenti informazioni: codice dell'utente che ha eseguito l'operazione, tipo di operazione effettuata (modifica o cancellazione del record), data in cui si è svolta l'operazione.

Per poter far uso di tale funzionalità è necessario attenersi ad alcuni vincoli nella costruzione delle relative tabelle del DB, le quali dovranno contenere obbligatoriamente i seguenti campi:

- T_NOMETABELLA_OID, campo numerico e chiave primaria;
- ID, campo numerico;
- REC_STATO: stato del record, il quale può essere attivo (A), modificato (U) o eliminato (D);
- REC_TSINIZIO: data di acquisizione dello stato A del record;
- REC_TSFINE: data di acquisizione dello stato U o D; dunque REC_TSINIZIO e REC_TSFINE delimitano l'intervallo di validità del record.
- REC_V_UTENTE_OID: codice dell'utente che ha eseguito l'operazione di insert/modifica/delete.

Per chiarire il meccanismo della cancellazione logica, si consideri il seguente esempio. La tabella T_TESTO contiene le informazioni relative ad alcune opere letterarie (titolo, descrizione, data stampa) ed è stata strutturata per supportare la storicizzazione:

T_TESTO {T_TESTO_OID, ID, T_TESTO_TITOLO, T_TESTO_DESCRIZIONE,
T_TESTO_DATA_STAMPA, REC_STATO, REC_TSINIZIO, REC_TSFINE,
REC_V_UTENTE_OID}

Si consideri ora il seguente record nella tabella:

10	3	Emma	Emma – Jane Austen	01/02/2011	A	18/10/2018	null	1
----	---	------	--------------------	------------	---	------------	------	---

Tale istanza, la numero 10, si riferisce al record 3, il quale è stato inserito in data 18/10/2018 dall'utente 1 e risulta attivo (A).

Un'operazione di modifica di questo record, da parte dell'utente 2, ad esempio al fine di correggere l'anno della data di stampa, produrrà:

- una modifica dello stato dell'istanza precedente, da A (attiva) ad U (modificata), con conseguente aggiornamento del campo REC_TSFINE;

- l'inserimento di una nuova istanza, attiva, riferita al record 3:

10	3	Emma	Emma – Jane Austen	01/02/11	U	18/10/18	19/10/18	1
11	3	Emma	Emma – Jane Austen	01/02/12	A	19/10/18	null	2

Come già accennato, per creare attraverso il BMF un oggetto di tipo *input-form* capace di supportare il data entry con storicizzazione, dopo aver correttamente strutturato la tabella del DB, sarà sufficiente scegliere per l'attributo *archivia* il valore S:

```
{
  "type": "inputForm",
  "inputForm": {
    "title": "",
    "tableName": "T_TESTO",
    "pk": { "auto": "S", "fields": ["T_TESTO_OID"] },
    "archivia": "S",
    "fields": [
      {
        "id": "T_TESTO_OID",
        "name": "T_TESTO_OID",
        "label": "ID",

```

```
"enable":"N",
"type":"text"
},
{
  "id":"T_TESTO_TITOLO",
  "name":"T_TESTO_TITOLO",
  "label":"Titolo",
  "type":"text",
  "validation":{"required": "true"}
},
{
  "id":"T_TESTO_DESCRIZIONE",
  "name":"T_TESTO_DESCRIZIONE",
  "label":"Descrizione",
  "type":"text",
  "validation":{"required": "true"}
},
{
  "id":"T_TESTO_DATA_STAMPA",
  "name":"T_TESTO_DATA_STAMPA",
  "label":"Data Stampa",
  "type":"text",
  "typeObj":"Date"
}}}
```

È possibile notare come i campi ID, REC_STATO, REC_TSINIZIO, REC_TSFINE, REC_V_UTENTE_OID vengano gestiti automaticamente e non debbano essere presenti tra i fields dell'*input-form*.

5.2.9 Data entry per riga

L'operazione di *data entry per riga* richiede che la tabella del DB su cui si vuole effettuare questo tipo di inserimento sia caratterizzata dalla seguente struttura:

```
T_TABLE_ID
T_TABLE_KEY
T_TABLE_VALUE
```

ovvero che in essa vi sono più record a comporre un dato.

Ad esempio, se il dato è costituito dai campi NOME (Mario), COGNOME (Rossi) e SESSO (M) di un individuo, nella tabella strutturata per riga questo dato sarà registrato attraverso l'inserimento di tre record:

T_TABLE_ID	T_TABLE_KEY	T_TABLE_VALUE
1	COGNOME	Rossi
1	NOME	Mario
1	SESSO	M

L'oggetto BMF che gestisce questa funzionalità è un oggetto di tipo *input-form* caratterizzato dalla seguente configurazione Json:

```
{
  "type": "inputForm",
  "inputForm": {
    "title": "Data entry per riga",
    "tableName": "T_TEST_RIGA",
    "type": "ROW",
    "pk": {"auto": "S", "fields": ["T_TEST_RIGA_ID"]},
    "fk": ["V_UTENTE_CODICE"],
    "fields": [
      {"id": "T_TEST_RIGA_ID",
       "name": "T_TEST_RIGA_ID",
       "label": "ID",
       "enable": "N",
```

```
"session": "N",
"type": "text"
},
{"id": "COGNOME",
"name": "COGNOME",
"label": "Cognome",
"session": "N",
"type": "text",
"validation": {"required": "true"}
},
{"id": "NOME",
"name": "NOME",
"label": "Nome",
"session": "N",
"type": "text",
"validation": {"required": "true"}
},
{"id": "SESSO",
"name": "SESSO",
"label": "Sesso",
"session": "N",
"type": "select",
list: [ {"val": "M", "descr": "M"}, {"val": "F", "descr": "F"} ],
"validation": {"required": "true"}
},
{"id": "V_UTENTE_CODICE",
"name": "V_UTENTE_CODICE",
"label": "utente",
"session": "N",
"type": "hidden",
"select": {"idObject": "S_UTENTE"}
}
],
"customization": "",
```

```
        "buttons":[{ "id": "btn_save",
"name": "btn_save", "label": "Salva", "iconClass": "glyphicon glyphicon-floppy-
saved"},{ "id": "btn_delete", "name": "btn_delete", "label": "Elimina"}]
    }
}
```

In particolare, per utilizzare la funzionalità di data entry per riga è necessario:

- assegnare all'attributo *type* il valore *row*;
- specificare eventuali *foreign key* in modo da gestirle correttamente in fase di insert e update.

Esempio InputForm Per Riga

ID	Cognome
21	Woodhouse
23	Smith
24	Churchill



ID

24

Cognome

Churchill

Nome

Frank

Sesso

M

Clear

Salva 

Elimina

Figura 33 – Esempio data entry per riga

5.3 Oggetti per la ricerca

Per eseguire ricerche sui dati del DB, il BMF mette a disposizione gli oggetti *filter*. I filtri, nel loro utilizzo standard, sono abbinati ad oggetti di tipo *report*, contenitori dei risultati della ricerca effettuata. Più precisamente, il filtro specifica la struttura dei parametri di ricerca ed ha il solo scopo di permettere all'utente di inserire i dati che saranno usati come parametri, ma come questi dati vengano poi utilizzati è specificato nella query del report abbinato.

5.3.1 Filter

L'oggetto di tipo Filter viene dunque utilizzato per definire la struttura dei parametri in base ai quali saranno filtrati i dati da visualizzare.

Step 1 - Parametri di configurazione

ID*: identificativo numerico dell'oggetto, settato automaticamente dal BMF;

Nome*: nome del filtro (per convenzione, il nome degli oggetti di tipo filtro inizia con "F_");

Tipo*: *filter*;

Descrizione: descrizione dell'oggetto;

DB: se non specificato, il DB di default del framework.

Anche per gli oggetti di tipo filtro i campi relativi ai parametri Link, LinkParams, Ordine non hanno significato. I campi contrassegnati con l'asterisco sono quelli obbligatori.

Step 2 – Json

Si partirà ancora una volta da un esempio: la configurazione del filtro mostrato in Figura 34. La parte evidenziata in blu rappresenta la struttura comune a tutti gli oggetti e l'attributo *type* indica che si sta configurando un filtro; l'attributo *filter* invece contiene le opzioni di configurazione proprie del relativo oggetto.

```

{"type": "filter",
"filter": {
  "title": "Ricerca in anagrafica",
  "dynamic": 'O',
  "fields": [
    { "id": "P_id",
      "name": "T_ANAG_ID",
      "label": "ID Paziente",
      "default": "", "type": "text",
      "operator": "icontains", "session": "N"},
    { "id": "P_cognome",
      "name": "T_ANAG_COGNOME",
      "label": "Cognome", "default": "",
      "type": "text",
      "operator": "icontains",
      "session": "N" },
    { "id": "P_nome",
      "name": "T_ANAG_NOME",
      "label": "Nome", "default": "",
      "type": "text",
      "operator": "icontains",
      "session": "N" },
    "hidden": "S",
    "buttonText": "Ricerca"  }}

```

Ricerca in anagrafica

ID Paziente

Cognome

Nome

Ricerca

Figura 34 - Esempio oggetto Filter

Le opzioni comprendono (quelle contrassegnate dall'asterisco sono quelle obbligatorie):

- **fields***: lista dei campi che compongono il filtro, per i quali è possibile specificare i seguenti attributi:

- **id**: identificativo html univoco (il valore di default è uguale a `<name>_id`);
- **name***: nome da assegnare al campo, utilizzato per comporre la query;
- **label**: etichetta descrittiva del campo (il valore di default è uguale a `<name>_id`);
- **type***: tipo del campo, a scelta tra:
 - **select** (il valore è selezionabile all'interno di una lista),
 - **hidden** (il campo è nascosto, non viene visualizzato, e il suo valore è quello di default),
 - **text** (testo libero);
 - **radio**;
 - **checkbox**;

(valgono le stesse considerazioni fatte per le modalità di inserimento dati negli oggetti di tipo *input-form* – pagina 54);

- **session**: indica se il parametro deve essere inserito tra quelli in sessione;
- **default**: valore di default;
- **placeholder**: valore visualizzato all'interno del campo fin quando questo rimane vuoto;
- **operator**: attributo che indica l'operatore da utilizzare nella composizione della condizione della query (valido solo se *session=false*). Può assumere i seguenti valori: `contains`, `equal` (default), `starts`, `ends`, `icontains`, `iequal`, `istarts`, `iends`, `>`, `<`, `>=`, `<=`. Gli operatori `contains`, `equal`, `starts`, `ends` sono case sensitive a differenza di `icontains`, `iequal`, `istarts`, `iends`;

- **dynamic** (S/N): scegliendo per questo attributo il valore S, attraverso una serie di checkbox sarà possibile decidere in real time quali colonne visualizzare nel report che si sta andando a generare (solo nel caso in cui vi sia un unico report nella pagina); se nessuna checkbox viene selezionata, tutte le colonne del report sono visualizzate.

- **collapse** (S/N): scegliendo per questo attributo il valore S, il filtro si chiuderà automaticamente dopo l'operazione di submit (ma sarà possibile espanderlo nuovamente per eseguire una nuova ricerca);

- **hidden** (S/N): scegliendo per questo attributo il valore S, il filtro sarà completamente nascosto a seguito dell'operazione di submit e non potrà essere riespanso;
- **buttonText**: label del pulsante ricerca.

Step 3 – Query Nel caso degli oggetti di tipo filtro, non è necessario scrivere codice SQL.

5.4 Oggetti per la navigazione

Gli oggetti per strutturare la navigazione all'interno del sito comprendono sia i pulsanti per la costruzione del menu di navigazione laterale (oggetti *menu item* e *menu folder*) sia i bottoni (*buttons* e *group buttons*), presenti direttamente all'interno delle maschere della web-app per migliorarne la fluidità logica.

5.4.1 Menu item e menu folder

Questi oggetti permettono di strutturare la navigazione del sito, creando il menu laterale, il quale, nell'aspetto standard delle applicazioni BMF, si trova al margine sinistro delle maschere della web-app.

L'oggetto di tipo *menu item* rappresenta un vero e proprio pulsante all'interno del menu di navigazione del sito: avrà pertanto un link associato, invocato dal click sul pulsante stesso. Al contrario, gli oggetti di tipo *menu folder* sono "contenitori" di oggetti di tipo *menu item* o di altri *menu folder*, per la loro organizzazione gerarchica, e perciò non hanno link associati; per il resto, la configurazione dei due oggetti è la stessa.

Step 1 - Parametri di configurazione

ID*: identificativo numerico dell'oggetto, settato automaticamente dal BMF;

Nome*: nome dell'oggetto (per convenzione, il nome degli oggetti di tipo menu inizia con "P_");

Tipo*: menuFolder/menuitem;

Descrizione: descrizione dell'oggetto;

Link: nel caso di oggetti menu item, è il link invocato (esempio: "link:{"#dispatcher/idObject=F_RICERCA&idObject=V_ANAGRAFICA&idObject=T_ANAGRAFICA"}");

DB: pool su cui verrà eseguita la query; se non specificato, il DB di default del framework;

Ordine: numero che indica la posizione dell'oggetto all'interno del suo menu-contenitore.

Step 2 – Json

La configurazione del file Json è davvero semplicissima: richiede soltanto la definizione dell'attributo *label*, etichetta da visualizzare; ad esempio:

```
{ "label" : "Dati Anagrafici" }
```

Step 3- Query

Non è necessario scrivere un'istruzione SQL.

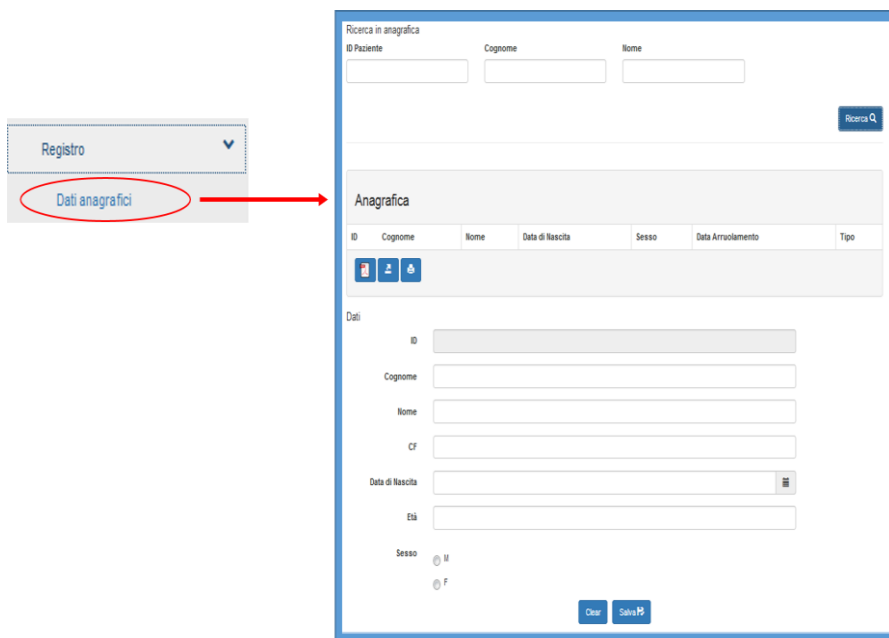


Figura 35 – Esempio Menu di navigazione: “Registro” è un oggetto di tipo menu folder, contenitore del pulsante “Dati anagrafici” (oggetto menu item) al click del quale viene invocato il link ad esso associato (che lancia la maschera in figura).

5.4.2 Button e Group button

Questi oggetti permettono di creare gruppi di pulsanti, utili per poter passare da una maschera all'altra della web-app, agganciando qualsiasi tipo di funzionalità.

Step 1 - Parametri di configurazione

ID*: identificativo numerico dell'oggetto, settato automaticamente dal BMF.

Nome*: nome dell'oggetto; per convenzione, il nome degli oggetti *group button* inizia con "GB_" mentre per gli oggetti di tipo *button* inizia con "BTN_";

Tipo*: *groupButton/button*;

Descrizione: descrizione dell'oggetto;

DB: pool su cui verrà eseguita la query; se non specificato, il DB di default del framework.

In questo caso, i campi Link, LinkParams, Ordine non hanno significato.

Step 2 – Json

Di seguito si riportano due esempi di configurazione, rispettivamente per *group buttons* e *buttons*. La parte evidenziata in blu rappresenta la struttura comune a tutti gli oggetti e l'attributo *type* indica che si sta configurando un bottone; gli attributi *group_buttons/buttons* contengono invece le opzioni di configurazione proprie dei relativi oggetti.

```
{"type": "group_buttons",
"group_buttons": {
  "items": [
    {'label': 'ECG', 'link': '#dispatcher/idObject=V_ECG'},
    {'label': 'EEG', 'link': '#dispatcher/idObject=V_EEG'},
    {'label': 'EMG', 'link': '#dispatcher/idObject=V_EMG'}
  ]
}
}
```



Figura 36 - Esempio oggetto Group button

```
{ "type": "buttons",  
  "buttons": {  
    "items": [{  
      'label': 'Nuovo Paziente',  
      'link': '#dispatcher/idObject=T_PAZIENTE&idObject=BTN_ESEMPIO'},  
      {  
        'label': 'Ricerca Paziente',  
        'link': '#dispatcher/idObject=F_PAZIENTE&idObject=V_PAZIENTE&  
idObject=BTN_ESEMPIO '}]  
    }  
  }  
}
```



Figura 37 – Esempio oggetto Button

Dunque, in entrambi i casi, editando il campo Json, è necessario definire la proprietà:

- *items*: lista dei singoli *item* ovvero dei singoli pulsanti che costituiscono l'oggetto.

Per ogni *item* devono obbligatoriamente essere definiti gli attributi *label* e *link*. Il primo indica l'etichetta del pulsante mentre il secondo è il link invocato al momento del click sullo stesso.

É necessario inserire almeno un elemento nel vettore *items*.

Step 3 - Query

Non è necessario scrivere istruzioni SQL.

5.5 Altri oggetti BMF

5.5.1 Action

L'oggetto di tipo *action* rappresenta un pulsante che ad oggi consente di implementare due diverse funzionalità: *help* e *print pdf*.

Help mette a disposizione un collegamento verso una pagina o un file di testo destinato a contenere informazioni aggiuntive utili all'utente (ad esempio una guida in linea). *Print pdf* genera un file pdf contenente l'oggetto *report* presente all'interno della pagina in cui l'oggetto *action* è stato inserito.

Step 1 - Parametri di configurazione

ID*: identificativo numerico dell'oggetto, settato automaticamente dal BMF;

Nome*: nome dell'oggetto (per convenzione, il nome degli oggetti di tipo *action* inizia con "A_");

Tipo*: action;

Descrizione: descrizione dell'oggetto;

DB: pool su cui verrà eseguita la query; se non specificato, è il DB di default del framework.

Per gli oggetti di tipo action i parametri Link, LinkParams, Ordine non hanno significato. I campi contrassegnati con asterisco sono quelli obbligatori.

Step 2 – Json

Partiamo da un esempio:

```
{"type":"action",  
"action":{  
  "type": "help",  
  "href":"help.htm",  
  "icon": "glyphicon glyphicon-question-sign"}}
```

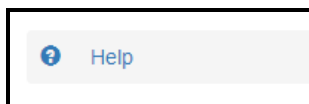


Figura 38 - Esempio oggetto Action Help

La parte evidenziata in blu rappresenta la struttura comune a tutti gli oggetti e l'attributo *type* indica che si sta configurando un oggetto *action*; l'attributo *action* contiene invece le opzioni di configurazione proprie del relativo oggetto, che per la funzionalità help comprendono:

- *type*: funzionalità implementata;
- *href*: nome della risorsa da visualizzare, inserita all'interno della *directory help*;
- *icon*: icona del pulsante.

Per la funzionalità *Print pdf*, invece, non è necessario specificare il nome di una risorsa da invocare:

```
{"type":"action",  
"action":{  
    "type": "print",  
    "icon": "glyphicon glyphicon-print"  
}}
```



Figura 39 - Esempio oggetto Action Print Pdf

Step 3 - Query

Per nessuna delle due funzionalità è necessario scrivere un'istruzione SQL.

5.5.2 Carousel

L'oggetto di tipo *carousel* consente la creazione di immagini statiche o di *slideshow*.

Step 1 - Parametri di configurazione

ID*: identificativo numerico dell'oggetto, settato automaticamente dal BMF;

Nome*: nome dell'oggetto (per convenzione, il nome degli oggetti di tipo *carousel* inizia con "C_");

Tipo*: carousel;

Descrizione: descrizione dell'oggetto;

DB: pool su cui verrà eseguita la query; se non specificato, è il DB di default del framework.

Per gli oggetti di tipo *carousel* i parametri Link, LinkParams, Ordine non hanno significato. I campi contrassegnati con asterisco sono quelli obbligatori.

Step 2 – Json

Partiamo da un esempio:

```
{"type": "carousel",  
  "carousel": {  
    "id": "C_CAROUSEL",  
    "images": ["banner_1.jpg", "banner_2.jpg"]}}
```

La parte evidenziata in blu rappresenta la struttura comune a tutti gli oggetti e l'attributo *type* indica che si sta configurando un oggetto *carousel*; l'attributo *carousel* contiene invece le opzioni di configurazione proprie del relativo oggetto, che comprendono:

- *id*: identificatore univoco dell'oggetto;
- *images*: lista dei file da inserire nella *slideshow* (se la lista contiene un solo file, sarà creata un'immagine statica) e contenuti all'interno della directory *images*.

Step 3 - Query

Per gli oggetti *carousel* non è necessario scrivere istruzioni SQL.

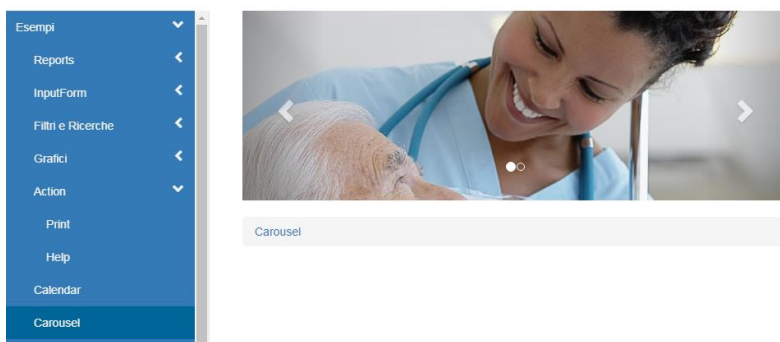


Figura 40 - Esempio oggetto Carousel

5.5.3 Period

Questo oggetto consiste semplicemente in un titolo (di default evidenziato in celeste) tramite il quale viene specificato il periodo di validità dei dati oggetto della reportistica.

La configurazione del campo Json per questo oggetto è molto semplice, come mostrato nell'esempio seguente:

```
{
  "type": "period",
  "period": {
    "title": "Dati aggiornati al Gennaio 2012 "
  }
}
```

Title è un attributo obbligatorio e contiene il messaggio che si intende mostrare per chiarire il periodo di validità dei dati oggetto di visualizzazione nella relativa maschera della web-app.

Nel caso di report annidati non sarà necessario passare l'oggetto in tutte le url ma solo nella chiamata del report più esterno.

Dati aggiornati al Gennaio 2018

Dizionario Edifici	
Id	1
Codice	POLO A
Id	2
Codice	POLO B
Id	3
Codice	POLO C
  	

Figura 41 - Esempio oggetto Period

5.5.4 Stored Procedure

Questo oggetto invoca una *stored procedure* implementata in uno dei DB configurati.

Per convenzione, il nome degli oggetti *stored procedure* inizia con "SP_".

La configurazione del campo Json è molto semplice: richiede la sola valorizzazione dell'attributo *type*:

```
{  
  "type": "storeproc",  
  "storeproc": { }  
}
```

Nel campo Query, sarà sufficiente richiamare il nome con cui la *stored procedure* è stata salvata nel DB, passando eventuali parametri in ingresso seguendo il formalismo:

- <nomeParametro>: nel caso si tratti di un parametro memorizzato in sessione;
- '<nomeParametro>': nel caso in cui il parametro non sia in sessione e venga passato manualmente all'interno della query string da eseguire.

Esempio:

1. P_ALBERO_ELIMINA_NODO(<P_idNODO>)

Lanciando il link:

#dispatcher/idObject=P_ALBERO_ELIMINA_NODO

sarà eseguita la stored procedure che elimina il nodo con codice P_idNODO salvato in sessione;

2. P_ALBERO_ELIMINA_NODO_2('<P_idNODO>')

Lanciando il link:

#dispatcher/idObject=P_ALBERO_ELIMINA_NODO&name1=P_idNODO&value1=5

sarà eseguita la stored procedure che elimina il nodo con codice 5.

5.5.5 Select

L'oggetto di tipo *select* permette di memorizzare una query SQL, ad esempio per utilizzarla al fine di valorizzare i campi di tipo *select*/*radio*/*checkbox* di un filtro o di un *input-form*.

Step 1 - Parametri di configurazione

ID: identificativo numerico dell'oggetto, settato automaticamente dal BMF;

Nome: nome dell'oggetto (per convenzione, il nome degli oggetti di tipo *select* inizia con "S_");

Tipo: *select*;

Descrizione: descrizione dell'oggetto;

DB: pool su cui verrà eseguita la query; se non specificato, il DB di default del framework.

Per gli oggetti di tipo *select* i parametri *Link*, *LinkParams*, *Ordine* non hanno significato.

Step 2 - Json

L'oggetto *select* non ha una parte *Json*, in quanto in uso solo lato server.

Step 3 - Query

Query rappresenta il cuore dell'oggetto, contenente la *SELECT SQL* che si intende memorizzare.

5.6 Opzioni avanzate

5.6.1 Record selezionabili: report annidati e associati ad input-form

Il BMF3 mette a disposizione dello sviluppatore la possibilità di rendere selezionabili i record di un oggetto *report*, ad esempio per l'invocazione di nuovi *report* o *grafici* (funzionalità *report* annidati) o per il caricamento di dati all'interno di oggetti di tipo *input-form* (*report* associati ad *input-form*).

Per sfruttare questa funzionalità, è necessario agire durante lo step 3 di configurazione dell'oggetto *report*: nella scrittura del codice SQL: in corrispondenza del campo che, per ogni record, si vuole rendere selezionabile, basterà specificare che dovrà trattarsi di un collegamento ipertestuale, ovvero basterà introdurre il tag HTML *a*, il cui attributo *href* conterrà il link da invocare. Dunque, non solo *report* annidati o associati a *input-form*: grazie all'attributo *href* sarà in realtà possibile invocare un qualsiasi link.

Esempio: Report annidati

```
select
RIC.T_RICOVERO_ID AS ID,
'<a
href="#dispatcher/idObject=V_Pazienti&name1=T_PAZIENTE_ID&value1='|
|RIC.T_PAZIENTE_ID|'>'|RIC.T_PAZIENTE_ID|'</a>' AS T_PAZIENTE_ID,
REP.T_REPARTO_NOME,
TO_CHAR (RIC.T_RICOVERO_DATA_INIZIO) AS DATA_INIZIO,
TO_CHAR (RIC.T_RICOVERO_DATA_FINE) AS DATA_FINE,
from T_RICOVERO RIC
left join T_REPARTO REP on REP. T_REPARTO_ID= RIC. T_REPARTO_ID
order by RIC.T_RICOVERO_ID
```

Cliccando sul campo ID di un record della tabella relativa ai ricoveri, viene invocato il report V_Pazienti con i dati anagrafici del paziente a cui si riferisce il ricovero selezionato.

Esempio: Report e input-form associati

```
select
'
```

Test Genetici - Whole Exome Sequencing

ID Record	Mutazione Individuata
2	SI

ID TEST

2

Mutazioni individuate

SI

Mutazioni individuate - descrizione

Missense

SI

Nonsense

No

Splicing

No

Frameshift

No

Note

Prova Test Genetico

Clear

Salva

Figura 42 –Esempio di report e input-form associati cliccando sul campo ID di un record in tabella, i relativi dati vengono caricati nell'input-form associato

5.6.2 Composizione link

I link predefiniti, cioè che scatenano eventi gestiti dal framework, sono di due tipi:

- **#dispatcher/...** : genera la pagina inizializzando tutti gli oggetti specificati.

Gli attributi ammessi sono:

- **idObject**: indica l'oggetto da caricare; in una stessa pagina possono essere caricati più oggetti elencando nella query string più *idObject* concatenati con il simbolo & (a condizione che siano uno per tipologia, ad eccezione degli oggetti di tipo *report*, *chart* e *calendar*, per i quali è prevista la visualizzazione multipla);
- **page**: nel caso si stia caricando un *report*, questo attributo (se uguale a 1) permette di attivare l'opzione di paginazione;
- **numRecPage**: nel caso in cui sia attiva l'opzione di paginazione, questo attributo imposta il numero di record per pagina;
- **clearParams** (S/N): attributo che gestisce lo svuotamento della sessione;
- **tree** (S/N): attributo che gestisce la generazione degli alberi.

Oltre a questi attributi, sono ammessi anche tutti quelli necessari alla composizione manuale di query string e alla gestione dei sottotitoli (paragrafo 5.6.3).




Esempio:

```
#dispatcher/idObject=V_Esempio_PerInputForm&idObject=T_EsempioInputForm&page=1&numRecPage=1
```

In questo caso, la pagina viene generata inizializzando l'oggetto report *V_Esempio_PerInputForm* e l'oggetto *input-form* *T_EsempioInputForm*.

Pazienti

Id	Nome	Cognome	CF	Sesso	Data di Nascita
2	LUIGI	VERDI	CFVERDILUIGI	M	15-12-2015
3	SIMONA	GIALLINI	CFGIALLINISIMONA	F	04-12-2008
1	MARIO	ROSSI	CFROSSIMARIO	M	25-12-2015
4	CARLO	BLU	CFBLUCARLO	M	11-10-1996
24	PALLO	PINCO	CFPINCOPALLO	M	18-10-2004



First

Previous

1

Next

Last

Esempio input-form


ID

Cognome

Nome

CF

Data di Nascita



Sesso

☐ M

☐ F

Clear


Salva 

Figura 43 – Esempio di maschera invocata dal link
`#dispatcher/idObject=V_Esempio_PerInputForm&idObject=T_EsempiInputForm&page=1`
`&numRecPage=1`

- **#loadRecord/...** : invoca la funzionalità di caricamento dei dati relativi al record selezionato all'interno dell'input-form.

Gli attributi ammessi sono:

- **idObject**: indica l'oggetto input-form nel quale si andrà a caricare il record selezionato.

Oltre a questi attributi, sono ammessi anche tutti quelli necessari alla composizione manuale di query string e alla gestione dei sottotitoli (paragrafo 5.6.3).

Esempio:

`#loadRecord/idObject=T_Anagrafe&name1=T_ANAG_ID&value1=3`

L'attributo idObject si riferisce all'oggetto di tipo input-form (T_Anagrafe) in cui saranno caricati i dati.

Gli attributi di query string, per filtrare i dati e recuperare il record voluto, sono:

- name: è il nome del campo della tabella del DB che agirà da filtro;
- value: è il valore del campo.

Dunque, nell'input-form T_Anagrafe saranno caricati i dati relativi al record della tabella DB T_ANAGRAFE in corrispondenza del quale il campo T_ANAG_ID vale 3. Modificando nel modo seguente il precedente esempio:

#loadRecord/idObject=T_Anagrafe&name1=T_ANAG_ID&value1='|T_ANAG_ID|'
il parametro usato da filtro non avrà valore costante, ma dinamico e uguale al valore dell'elemento HTML con nome T_ANAG_ID.

Dunque, associando questa tipologia di link al parametro href di un campo di un oggetto *report*, cliccando sul quel campo verrà invocato l'input-form e compilato automaticamente con i dati relativi allo specifico record selezionato.

The screenshot shows a web application interface titled "Anagrafica". At the top, there is a table with 7 columns: ID, Cognome, Nome, Data di Nascita, Sesso, Data Arruolamento, and Tipo. The table contains one record with ID 24, Cognome BIANCHI, Nome GIGI, Data di Nascita 01-01-1978, Sesso M, Data Arruolamento 14-03-2018, and Tipo R. Below the table are three icons (document, print, and another document). Underneath these icons is a pagination bar with buttons for "First", "Previous", "1" (selected), "2", "3", "4", "5", "6", "7", "8", "Next", and "Last". Below the pagination bar is a section titled "Dati" containing several input fields: "ID" (with value 24), "Cognome" (with value BIANCHI), "Nome" (with value GIGI), "CF" (with value CFBIANCHIGIGI), "Data di Nascita" (with value 01-01-1978 and a calendar icon), "Età" (with value 40), and "Sesso" (with radio buttons for M and F, where M is selected). At the bottom of the form are two buttons: "Clear" and "Salva" (with a save icon).

Figura 44 - Caricamento automatico dei dati nell'input-form, dopo la selezione del campo ID nel report Anagrafica

5.6.3 Composizione della query string e gestione dei sottotitoli

Oltre all'elenco degli idObject ovvero degli oggetti con cui popolare la maschera invocata, nella query string che compone il link potranno essere specificati i parametri di gestione delle condizioni SQL.

Ogni parametro viene definito da cinque attributi:

- **name***: nome del parametro (lo stesso nome che sarà utilizzato per riferirsi a tale parametro nelle istruzioni SQL);
- **value***: valore del parametro;
- **type**: tipo di dato (Date, Int, Double, String, Bool; String è il valore di default);
- **session (S/N)**: attributo che indica se il parametro deve essere o meno memorizzato in sessione (N è il valore di default);
- **operator**: operatore da utilizzare nella condizione SQL (= per default).

Esempi:

- **session1=S&name1=P_ANNO&value1=2010**

In sessione sarà memorizzato il parametro P_ANNO, il cui valore è 2010

- **name1=P_ANNO&value1=2010**

In questo caso il parametro sarà utilizzato per l'esecuzione della query, ma non sarà memorizzato in sessione.

Più parametri possono essere definiti in una stessa query string:

- **session1=S&name1=P_ANNO&value1=2010&&name2=codice_reparto&value2=123&type2=Int&name3=nome_prest&value3=ML123&operator3=<>**

Nel hashmap dei parametri "condizioni ricerca" sarà inserita la coppia (P_ANNO,2010); inoltre, sarà creata la condizione codice_reparto=? and nome_prest<> ? dove il primo parametro sarà gestito come Integer.

Nella query string possono essere inseriti anche gli attributi:

- **sqlConditione**: dà la possibilità di definire una specifica condizione SQL che verrà aggiunta alla query che si sta chiedendo di eseguire;
- **orderBy**: per l'applicazione dell'*order by* alla query; ad esempio, alla query string: `orderBy=codice_prest&orderBy=codice_reparto` corrisponderà l'istruzione SQL "Order by codice_prest,codice_reparto".

In questo modo, anche la gestione di report annidati risulta essere semplice e flessibile: possono essere composte condizioni di vario genere e possono essere aggiunti nuovi parametri in sessione passando da un report all'altro.

Infine, è all'interno delle query string che avviene la gestione dei sottotitoli da visualizzare nelle maschere di reportistica e/o data entry. In ogni query string possono essere inseriti da 0 a N sottotitoli, ognuno strutturato secondo le seguenti regole di configurazione:

- **tit_valueN** (dove N rappresenta il numero del sottotitolo): indica il valore del sottotitolo;
- **tit_labelN** (dove N rappresenta il numero del sottotitolo): indica l'etichetta del sottotitolo;
- **tit_keyN** (dove N rappresenta il numero del sottotitolo): indica la chiave del sottotitolo, obbligatoria quando si vuole utilizzare la rimozione dei sottotitoli attraverso l'opzione *tit_removeN* e quando la query string è contenuta in un link di loadRecord per il caricamento dei dati del *report* nel form di inserimento abbinato.

Inoltre:

- **tit_removeN** (dove N rappresenta il numero del sottotitolo): questo attributo sarà valorizzato con la chiave del sottotitolo che si desidera rimuovere;
- **tit_clear** (S/N): attributo che gestisce la cancellazione dei sottotitoli; il suo valore di default è true (S).

6. Personalizzare la web-app

Quale piattaforma open source, BMF3 permette allo sviluppatore di personalizzare il comportamento e l'aspetto della web-app, arricchendo gli oggetti standard di funzionalità aggiuntive, implementate direttamente lato *front-end* in linguaggio *JavaScript*.

Come già discusso in precedenza, la parte client del framework è completamente realizzata in *JavaScript*, con il supporto di librerie, tra le quali *Backbone* e *Marionette*, le quali offrono soluzioni architetturali per ottimizzare l'organizzazione e la strutturazione del codice; esse rientrano nella categoria delle librerie MV*, in quanto il loro approccio si basa sull'implementazione dei *Models* e delle *Views*, alle quali sono delegati anche i compiti del tradizionale componente *Controller*. Il *Model* è un oggetto discreto contenente una serie di dati sottoforma di attributi, mentre la *View* rappresenta il tramite fra interfaccia e modelli, definendone la logica di interazione: essa osserva i *models* e reagisce ai loro cambiamenti di stato.

Sebbene una buona conoscenza di questi strumenti sia necessaria al fine di comprendere a pieno le modalità di funzionamento del BMF (se ne raccomanda pertanto un opportuno approfondimento), si può intuire come, per modificare il comportamento degli oggetti BMF, sia necessario agire estendendo le *Views* standard con cui gli stessi vengono creati.

La struttura della directory di una web-app, realizzata mediante piattaforma BMF, comprende una cartella, *js*, deputata a raccogliere tutti i file *JavaScript* necessari al funzionamento dell'applicativo, suddivisi a seconda della tipologia di oggetto BMF a cui si riferiscono. È all'interno di questi file che sono implementati i *Models* e le *Views* per la realizzazione dei componenti standard della web-app.

Per creare le customizzazioni sarà necessario estendere le ***Views*** standard e dichiarare ed esporre le estensioni create all'interno del file ***js/customization/customization.js***, che raccoglie tutte le personalizzazioni.

Gli oggetti standard per i quali è prevista la possibilità di customizzazione sono: **report**, **filtri**, **input-form**. Andremo di seguito a presentare facili esempi di personalizzazione relativi a tutte e tre queste categorie, partendo dall'oggetto filtro.

6.1 Filtro personalizzato

La View che crea un oggetto filtro standard si chiama ***filtroItemView***, ed è esposta dallo script **filtro.js** nella directory *js/filtro*:

```
js/filtro/filtro.js
```

È estendendo questa View e sovrascrivendone proprietà e metodi che sarà possibile modificare il comportamento e/o l'aspetto del filtro; ad esempio:

```
var MyFilter = Filtro.filtroItemView.extend({
  events: function() {
    //Eredita gli events dalla view originale, dopodichè la estende
    Return _.extend(
      {},
      Filtro.filtroItemView.prototype.events,
      { "submitForm": "onFormSubmit" }
    );
  },
  onFormSubmit: function(){
    alert("Funzione filtro personalizzata");
    return false;
  }
});
```

Nell'esempio sopra riportato, è stata creata una *View* personalizzata, dal nome *MyFilter*, che estende la *View* standard *filtroItemView* e ne riscrive il metodo *submitForm* (invocato a seguito del click sul pulsante di ricerca) per visualizzare un semplice messaggio di alert a seguito dell'operazione di submit.

L'estensione creata dovrà essere dichiarata ed esposta all'interno del file *js/customization/customization.js*:

```
return { myFilter: MyFilter }
```

Per essere poi attribuita ad uno specifico componente di tipo filtro della nostra web-app, sarà necessario agire durante lo *step 2* della sua configurazione, ossia quello relativo alla scrittura del file *Json*, introducendo l'attributo *customization*:

```
"customization": "myFilter"
```

6.2 Input-form personalizzato

Si consideri ora l'oggetto *input-form*: la *View* per la creazione di un *form* standard si chiama ***inputformItemView***, e si trova nella directory:

```
js/inputform/inputform.js
```

Nell'esempio proposto viene creata una *View custom*, *MyInputForm*, estendendo *inputformItemView* e sovrascrivendo il metodo invocato al click del pulsante con id *btn_run*, ancora una volta per visualizzare un semplice messaggio di alert:

```
var MyInputForm = InputForm.inputformItemView.extend({
  events: function() {
    //Eredita gli events dalla view originale, dopodiché la estende
    return _.extend(
      {},
      InputForm.inputformItemView.prototype.events,
      {"click #btn_run": "myFunction"});
  },
  myFunction: function() {
    alert("Funzione input-form personalizzata");
  }
});
```

Anche in questo caso, l'estensione creata dovrà essere dichiarata ed esposta all'interno del file *js/customization/customization.js* che raccoglie tutte le personalizzazioni:

```
return { myInputForm: MyInputForm}
```

ed attribuita ad uno specifico oggetto *input-form* introducendo nel suo file *Json* di configurazione l'attributo *customization*:

```
"customization": "myInputForm"
```

6.3 Report personalizzato

Si mostra infine un semplice esempio di personalizzazione dell'oggetto *report*, al fine di modificarne l'aspetto grafico. La *View* per la creazione di un report standard è ***datiListView***, in cui *template* è l'attributo a cui è possibile associare il file *html* che definisce il *templating*:

```
var MyCustomHTML = Dati.dataListView.extend(  
    { template: CustomTemplates.myDati } );
```

myDati sarà pertanto un file *html*, all'interno della directory rappresentata dalla variabile *CustomTemplates*.

6.4 Altre Personalizzazioni

Tra i file presenti all'interno della cartella *js* si trova anche ***global.js***, nel quale sono definite alcune variabili quali:

- ***titleApp***, per scegliere il titolo dell'applicativo;
- ***ctx***, context dell'applicativo.

Il file ***favicon.ico***, nella cartella *images*, rappresenta l'icona del sito caricata nel *tab* del *browser*. Nella stessa cartella si trovano i file immagine per la realizzazione dei loghi, dei banner e degli oggetti *carousel* della web-app.

Per personalizzare l'aspetto della propria applicazione in termini di stile (font, colori, margini, disposizione degli elementi del dom, etc.) si potrà agire direttamente sui file *.css* raccolti nella cartella *css*, oppure si potrà utilizzare il seguente link:

<http://getbootstrap.com/customize/>

per una configurazione guidata (dove, caricando il file ***config.json***, sarà possibile partire dalle opzioni già impostate).

Scegliendo invece di intervenire manualmente sui file *.css* (ragionevole quando le modifiche da apportare sono minime), la soluzione raccomandata è quella di creare un nuovo file (ad esempio *myCustomStyle.css*) in cui inserire le modifiche da apportare (ricordando di aggiungere ***!important*** al termine di ogni istruzione al fine di sovrascrivere le impostazioni precedenti).

7. Upload e download file

Il BMF3 rende disponibili servizi, implementati lato server, al fine di caricare e scaricare files, per poter utilizzare i quali è innanzitutto necessario configurare, nel file di properties (***bmf3.properties***), la variabile ***path_upload***, indicante la directory nella quale si andranno a salvare e/o scaricare i files. È ovviamente necessario disporre dei diritti di scrittura/lettura sulla directory considerata.

Tali servizi vengono invocati tramite URL.

L'URL relativa alla funzionalità di download è ***downloadFile.bmf*** e necessita dei seguenti parametri:

- **name**: nome del file, con cui lo stesso è stato salvato nella directory;
- **nameOriginal**: nome con cui si vuole restituire il file, comprensivo di estensione;
- **subdir**: eventuale sottodirectory all'interno di *path_upload* dove recuperare il file;
- **contentDisposition**: parametro per definire la modalità di apertura del file (il valore *inline* è quello di default e permette di aprire il file in una pagina del browser, mentre il valore *attachment* permette di scaricare il file).

Di seguito, un esempio di URL per l'invocazione del servizio di download:

```
../config/downloadFile.bmf?name=ECG1&nameOriginal=ECG1.txt&subdir=ECG&contentDisposition=attachment
```

Per quanto riguarda invece la funzione di caricamento, l'URL è ***uploadFile.bmf***; di seguito un esempio nel quale la stessa viene invocata all'interno di una chiamata Ajax:

```
$.ajax({  
    url: "../config/uploadFile.bmf",  
    type: "POST",  
    data: formData,
```

```
cache: false,  
contentType: false,  
processData: false,  
success: function (data, textStatus, jqXHR){  
    alert ('Caricamento andato a buon fine');  
}}
```

L'oggetto FormData contiene le seguenti proprietà (quelle indicate con l'asterisco devono essere obbligatoriamente definite):

- **file***: il file caricato;
- **name***: il nome effettivo del file;
- **nameOriginal***: il nome con cui si vuole salvare il file, comprensivo di estensione;
- **subdir**: eventuale sottodirectory di path_upload nella quale salvare il file (può essere creata dinamicamente).

Per utilizzare tale funzionalità, inoltre, il template associato all'oggetto di tipo input-form utilizzato per il caricamento dovrà contenere un campo di tipo *type*, attraverso cui caricare il file di interesse.

8. Alberi

Il BMF3 mette a disposizione una serie di funzionalità al fine di creare particolari strutture gerarchiche di organizzazione dei dati: gli alberi.

L'Albero rappresenta infatti l'organizzazione gerarchica di un'azienda ed è costituito dalle singole strutture aziendali, a partire dalla Radice (Azienda) fino alle Foglie (es. ambulatori).

Nel caso in cui si vogliano ottenere "viste organizzative" diverse, è necessario creare alberi differenti; ad esempio, potremmo costruire un albero contenente solo le strutture di tipo sanitario, un altro con solo quelle di tipo amministrativo, oppure un misto delle due.

Definito un albero bisognerà definire un tipo albero per ogni tipologia di dati differenti che si vogliono rappresentare.

Avremo, ad esempio, un tipo albero ATTIVITA' (il nome è libero ed in genere è esplicativo dei dati rappresentati) per mostrare l'attività delle strutture associate, ed un tipo albero ATTRAZIONE per mostrare l'attrazione sulle stesse strutture.

Il menu BMF preposto alla gestione degli alberi è rappresentato in Figura 45. Di seguito andremo a descrivere le diverse voci visualizzate.



Figura 45 - Menu Alberi

8.1 Strutture

Come detto, un albero è costituito dalle diverse strutture aziendali, in un certo rapporto gerarchico l'una con l'altra. Il primo passo nella costruzione di un albero, pertanto, è la definizione di tali strutture. Per farlo, si utilizza la voce menu **strutture**, contenente le funzionalità per la creazione di nuove strutture e la gestione di quelle già create.

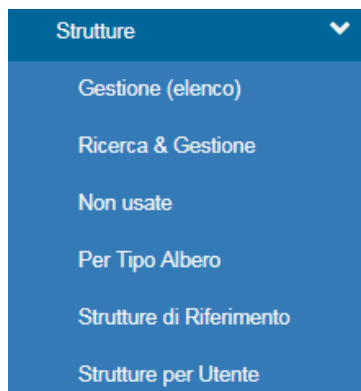


Figura 46 - Menu Strutture

Le funzionalità principali sono **Gestione (elenco)** e **Ricerca & Gestione**, attraverso le quali è possibile definire le strutture che comporranno gli alberi, oppure modificare/eliminare quelle già create. Per ogni struttura creata è obbligatorio definire la struttura di riferimento e il tipo struttura attraverso le rispettive combo.

Gestione Struttura Aziendale

Codice	Denominazione	Codice Aziendale	Codice Mnemonico	Codice SI	Struttura Tipo (CENTRI X VOCI)	Data Inizio (GG/MM/AAAA)	Data Fine (GG/MM/AAAA)	Struttura di Riferimento	Tipo Struttura
583									

First

Previous

1

2

3

4

5

6

7

8

9

10

Next

Last

Codice

Denominazione *

Codice Aziendale *

Codice Mnemonico

Codice SI

Struttura Tipo (CENTRI PER VOCI)

Data Inizio

Data Fine

Struttura di Riferimento *

Tipo Struttura *

Clear

Salva

Elimina

Figura 47 - Maschera di Gestione Strutture

Le altre funzionalità disponibili sono:

- **Non usate:** mostra l’elenco delle strutture create ma non utilizzate,
- **Per Tipo Albero:** filtra le strutture create per tipologia di albero di appartenenza (Figura 48);
- **Strutture di Riferimento:** definisce le strutture di riferimento.
- **Strutture per Utente:** filtra le strutture create per utente (Figura 49).

Strutture per Tipo Albero

Struttura (Ricerca in lista)

Struttura (Ricerca con autocompletamento)

Tipo Struttura

Tipo Albero

Link

Parametro

Ricerca

Per Tipo Albero / Ricerca

Strutture per Tipo Albero

Struttura (Cod.Alb. - Cod.Stru)	Tipo Albero	Link	Parametro
	1 --- ATTIVITA'		

First

Previous

1

2

3

4

5

6

7

8

9

10

Next

Last

Figura 48 - Maschera Strutture per Tipo Albero

Utenti & Strutture

Struttura

Tipo Struttura

Tipo Albero

Ricerca

Strutture per Utente / Ricerca

Utenti & Strutture

Utente	Albero	Tipo Albero
AMMINISTRATORE DI SISTEMA	10000 - ALBERO ATTIVITA'	ATTIVITA'

First

Previous

1

2

3

4

5

6

7

8

9

10

Next

Last

Figura 49 - Maschera Strutture per Utente

8.2 Radici

La voce Radici contiene le funzionalità per la definizione della radice di un albero, ovvero dell'elemento da cui l'albero si origina e che raggruppa le varie strutture che lo compongono. Di fatto, dunque, la radice è l'elemento identificatore dell'albero.






Figura 50 - Menu Radici

8.2.1 Definizione Radici

La prima funzionalità serve per definire le radici attraverso i campi **Codice Radice** e **Descrizione**, nella quale si suggerisce di ripetere il codice, in quanto tale campo viene richiamato in altre interrogazioni.

Gestione Radici Albero

Codice Interno	Codice Radice	Descrizione
1	10000	10000 - ALBERO ATTIVITA



First Previous **1** 2 3 4 5 6 7 8 9 Next Last

Codice Interno

1


Codice Radice *

10000

Descrizione *

10000 - ALBERO ATTIVITA

Clear

Salva 

Elimina

Figura 51 - Maschera definizione radici

8.2.2 Strutture Radici

Con questa seconda funzionalità si completa la definizione della radice, specificando quale tra le strutture create svolgerà tale ruolo: l'associazione Struttura - Radice avviene tramite le combo predefinite.

Associazione Struttura Radice

Codice	Struttura	Tipo	Codice Radice	Nome Albero	Codice Padre	Data Inizio	Data Fine
31295		RADICE	20000	20000 - ALBERO CUP		01-01-1990	

First

Previous

1

2

3

4

5

6

Next

Last

Codice Interno

31295

Struttura *

Radice*

20000 - ALBERO CUP

Clear

Salva

Elimina

Figura 52 - Maschera Struttura Radice

8.3 Tipi

Questa voce menu raccoglie le funzionalità per la definizione dei **tipi** da associare ad ogni albero ovvero in base alla tipologia di dati che si vogliono rappresentare utilizzando l'albero stesso. Come già anticipato, potremo avere, ad esempio, un tipo albero ATTIVITA' per la rappresentazione delle attività delle strutture componenti l'albero, ed un tipo albero ATTRAZIONE per mostrare l'attrazione sulle stesse strutture.

Tipi

Tipi

Duplica Tipo

Figura 53 - Menu Tipi

Tali funzionalità comprendono:

- **Tipi:** da utilizzare nel caso di una prima creazione di un Tipo albero, oppure quando non esiste un tipo albero simile a quello che vogliamo costruire; queste informazioni vengono memorizzate nella tabella T_ALBERO_TIPO.

- **Duplica Tipo:** una volta costruito un albero con un certo tipo albero, e definite le viste in associazione ai diversi nodi dell'albero (mediante Link per Struttura o Link puntuale), possiamo servirci della funzionalità Duplica tipo albero: otterremo automaticamente la duplicazione di quel particolare tipo albero, e l'associazione di tutti i nodi alla stessa vista specificata per il tipo albero d'origine. In figura, vengono mostrati i campi da specificare per questa utility: selezionare l'Albero di partenza, e indicare il Nuovo nome Tipo Albero.

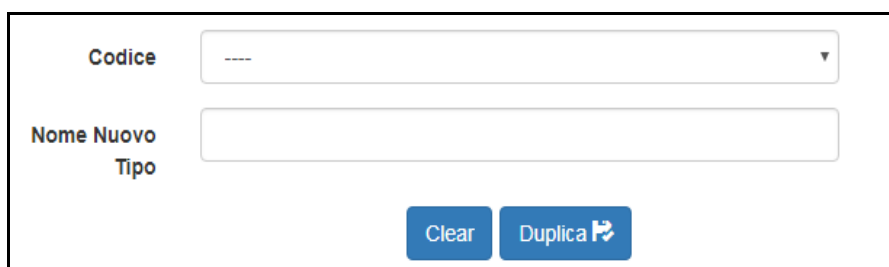


Figura 54 - Maschera Duplica Tipo

Cliccando su **Salva** verranno automaticamente effettuate le seguenti operazioni:

1. Inserimento di un nuovo record nella tabella **TIPO_ALBERO** con nome uguale a quello indicato nel campo Nuovo nome Tipo Albero;
2. Duplicazione dei record relativi al tipo albero origine nella tabella **T_ALBERO_LINK**.

8.4 Navigazione Gerarchie

A questo punto, dopo aver definito le strutture, individuato le radici e specificato i tipi, è possibile procedere alla creazione vera e propria dell'albero, assegnando le gerarchie, ovvero indicando i padri e/o i figli delle strutture all'interno dell'organizzazione gerarchica. A tale scopo viene utilizzata la funzionalità Navigazione Gerarchie, che permette in maniera visuale ed immediata la creazione e composizione di un albero.

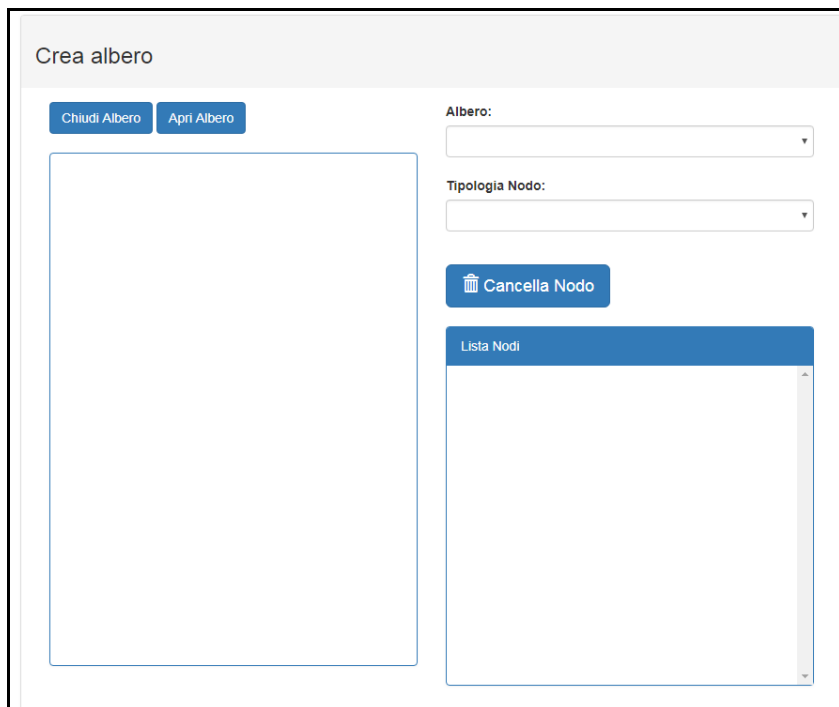


Figura 55 - Maschera Navigazione Gerarchie

Dal menu a tendina sulla destra della maschera, etichettato come Albero, viene scelta la radice da cui l'albero avrà origine. A seguito di tale selezione, sulla sinistra della maschera comparirà l'albero così come lo stiamo configurando.

Il tipo di nodo viene scelto attraverso il menu a tendina Tipologia Nodo; la lista sottostante permette di selezionare la voce di interesse e di trascinarla direttamente nell'albero: in questo modo il nodo selezionato e tutto il suo "sotto albero" sarà associato al livello in cui è stato trascinato.

8.5 Associa Link a Struttura

La funzionalità Associa Link a Struttura serve a definire l'azione da associare ai diversi nodi dell'albero, considerando i vari livelli di cui si compone. La maschera associata a questa funzionalità permette di selezionare l'Albero e il Tipo Albero da considerare attraverso le rispettive combo. A seguito di questa selezione, comparirà in basso l'albero "aperto" e si vedranno subito tutti gli elementi

dell'albero su cui c'è o andrà inserito un link (comando da eseguire) e gli eventuali parametri da passare.

L'azione associata a ciascun nodo è definita nel campo Link e permette di richiamare oggetti Report differenti per ciascun nodo.

Ovviamente tali oggetti Report devono essere stati precedentemente definiti.

Opzionalmente, sarà possibile definire il Parametro da passare al link invocato.

Per disattivare il link su una struttura, sarà possibile selezionare la struttura direttamente dall'albero in basso e premere il pulsante Elimina.

Per aggiornare il link di una o più strutture già associate, dopo aver selezionato la struttura dall'albero in basso, si andranno a modificare i campi link e/o parametro, confermando l'azione con il pulsante Salva Link e/o Salva Parametro.

Con il pulsante Salva tutto invece si potrà salvare tutto l'albero per quel link e quei parametri.

Si osservi che il campo Link non è obbligatorio. Se si vuole che il click su un nodo dell'albero non provochi alcuna azione, ad esempio un nodo di livello intermedio, su di esso non va definito il link (campo Link vuoto). In alternativa, al nodo in questione è possibile associare una pagina html statica (di default) con una spiegazione per l'utente su cosa deve fare. Questa pagina dovrà essere memorizzata nella directory dell'applicativo.

Le operazioni effettuate attraverso questa funzionalità vengono registrate nella tabella T_ALBERO_LINK.

Associa Link

Albero:
10000 - ALBERO ATTIVITA

Tipi di Albero:

Link:

Parametro:

Salva Tutto Salva Link Salva Parametro Cancella

Chiudi Albero Apri Albero

FGM

Figura 56 - Maschera Associa Link a Struttura

8.6 Controlli

La voce Controlli raggruppa tutte quelle funzionalità per verificare che non vi siano anomalie o incongruenze nella struttura degli alberi creati. Esse comprendono:

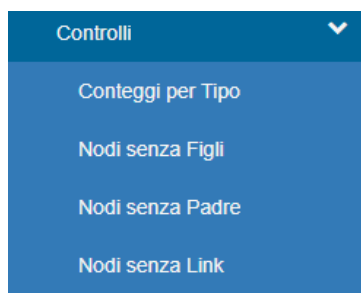


Figura 57 - Menu Controlli

- **Conteggi per Tipo:** interrogazione tramite cui è possibile verificare quanti record sono stati associati a ciascun tipo albero. E' così possibile verificare immediatamente le eventuali incongruenze, ad esempio tra tipi alberi che dovrebbero avere lo stesso numero di elementi;
- **Nodi senza Figli:** interrogazione tramite cui è possibile verificare se esistono dei nodi intermedi che erroneamente sono rimasti senza "figli", ovvero che non possono visualizzare alcuna attività in quanto pur essendo dei NODI non sono referenziati come padri di elementi gerarchicamente inferiori.
- **Nodi senza Padre:** interrogazione tramite cui è possibile verificare se esistono dei nodi intermedi che erroneamente sono rimasti senza "padre", ovvero che non possono essere visualizzati perché non sono richiamanti da alcun elemento gerarchicamente superiore.
- **Nodi senza Link:** interrogazione tramite cui è possibile verificare se esistono dei nodi che erroneamente sono rimasti senza "azione", ovvero senza link.

8.7 Sicurezza

La voce Sicurezza raggruppa quelle funzionalità necessarie alla gestione delle abilitazioni alla visualizzazione dei dati, da parte di un utente, per le singole strutture.

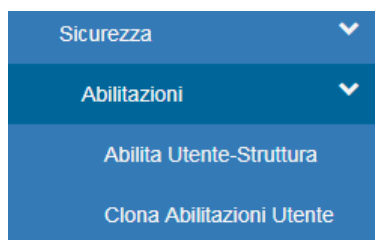


Figura 58 - Menu Sicurezza

Tali funzionalità comprendono:

- **Abilita Utente-Struttura:** abilita un singolo utente alle diverse strutture di sua competenza.

L'operatore selezionerà l'utente, l'albero di interesse e il livello di profondità.

L'albero di interesse è scelto mediante relativa combo, mentre per la selezione dell'utente si procede scrivendo le relative prime lettere nel text field "utente": il BMF caricherà in automatico la lista dei nomi, presa dalla V_Utente, per quegli utenti che soddisfano il parametro di ricerca (quei caratteri che si inizia digitare).

Il Livello massimo profondità (obbligatorio) esprime il livello di dettaglio con cui vogliamo visualizzare l'albero. Scegliendo ad esempio AMBULATORIO si vuole che per l'utente AMMINISTRATORE SITO l'albero venga visualizzato fino al livello Ambulatorio.

L'abilitazione si effettua spuntando le strutture di interesse nell'albero rappresentato in fondo alla maschera.

L'abilitazione si deve sempre eseguire fino alle strutture di livello FOGLIA, poiché solo queste strutture contengono i dati. Anche se l'utente in questione è ad esempio responsabile di area, dovremo scegliere tutte le strutture che appartengono a quell'area, ed associarle all'utente.

Abilita Utente a Struttura

Albero:

10000 - ALBERO ATTIVITA-ATTIVITA' ▼

Utente:

AMMINISTRATORE DI SISTEMA

Salva

Livello:

AMBULATORIO ▼

Chiudi Albero Apri Albero

- FGM-CREAS-IFC-CNR
 - Massa
 - Spec. Ambulatoriale MS
 - Cardiologia adulto
 - Biosegnali Adulto
 - Ecocardiografia Adulto
 - Cardiochirurgia pediatrica

Figura 59 - Maschera Abilita Utente - Struttura

- **Clona Abilitazioni Utente:** clona tutti i diritti di lettura/scrittura dell'utente abilitato nell'utente da abilitare. Scrivendo le prime lettere nel text field "utente da abilitare" il BMF caricherà in automatico la lista dei nomi, presa dalla V_Utente, per quegli utenti che soddisfano il parametro di ricerca (quei caratteri che si inizia digitare). Se l'utente abilitato non esiste non si può clonare alcun diritto, stesso discorso vale per l'utente da abilitare. Questa funzione esegue correttamente la clonazione solo nel caso in cui esistano i due utenti.

I dati registrati attraverso le due funzionalità vengono memorizzati nella tabella **REL_UTENTE_ALBERO**.

Figura 60 - Maschera Clona Abilitazioni Utente

8.8 Database e Viste V_ALBERO_*

La navigazione degli alberi personalizzati è gestita lato DB grazie alle **Viste** sotto elencate, generate attraverso l'esecuzione di specifici **script SQL**, che contengono i dati relativi alla rappresentazione degli alberi di navigazione:

V_ALBERO

V_ALBERO_LIVELLI_2

V_ALBERO_LIVELLI_3

V_ALBERO_LIVELLI_4

V_ALBERO_LIVELLI_5

V_ALBERO_LIVELLI_6

V_ALBERO_LIVELLI

V_ALBERO_COMPLETO

V_ALBERO_ELEM

V_ALBERO_LINK

V_ALBERO_NUMERO_RECORD

V_ALBERO_RADICI_TIPO

V_ALBERO_MAPPA

Queste viste verranno utilizzate in join con le tabelle dell'applicativo, per la costruzione della specifica query (da definire nell'oggetto **Vista** dell'applicativo) che visualizzi i dati e l'albero di navigazione.

L'utente non dovrà mai modificare tali viste, per non pregiudicare il funzionamento del sistema.

8.9 Oggetti Tree

Dopo aver predisposto l'organizzazione gerarchica attraverso le funzionalità sopra descritte, sarà possibile associare tale organizzazione ad un particolare oggetto BMF: tree.

La configurazione di questo oggetto è molto semplice:

- **Step 1: Parametri di configurazione**

ID*: identificativo numerico dell'oggetto, settato automaticamente dal BMF;

Nome*: nome dell'oggetto (per convenzione, il nome degli oggetti *input-form* inizia con "TREE_");

Tipo*: tree;

Descrizione: descrizione dell'oggetto;

DB: pool su cui verrà eseguita la query (se non specificato, il DB di default del framework).

I campi relativi ai parametri Link, LinkParams, Ordine non hanno significato per gli oggetti di tipo *tree*. I campi contrassegnati con l'asterisco sono quelli obbligatori.

- **Step 2: Json**

Si riporta di seguito un esempio di configurazione del campo Json per questi particolari oggetti. L'attributo *type* indica che si sta configurando un oggetto tree mentre l'attributo *tree* contiene le opzioni di configurazione proprie di questo particolare oggetto.

```
{"type": "tree",  
  "tree": {  
    "radice": "20000",  
    "tipo": "6"}}
```

Tali proprietà comprendono:

- **radice***: indica il codice della radice dell'albero che vogliamo associare all'oggetto tree;
- **tipo***: indica il codice del tipo albero che si intende associare all'oggetto tree.

- **Step 3: Query**

Per questa tipologia di oggetti non è necessario scrivere una query SQL.